Refine Search

Your wildcard search against 10000 terms has yielded the results below.

Your result set for the last L# is incomplete.

The probable cause is use of unlimited truncation. Revise your search strategy to use limited truncation.

Search Results -

Terms	Documents
L10 and (database same ind$)	1

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L16

Refine Search

Recall Text | Clear | Interrupt

Search History

DATE: Tuesday, February 24, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
	DB=USPT; THES=ASSIGNEE; PLUR=YES; OP=OR		
L16	L10 and (database same ind$)	1	L16
L15	L10 and (search$ same ind$)	1	L15
L14	L10 and (search$ same database same ind$)	0	L14
L13	L10 and (micro$ or macro$) and (search$ same database same ind$)	0	L13
L12	L10 and (micro$ or macro$) and (search$ same database same index)	0	L12
L11	L10 and (micro$ or macro$) and (search$ with database with index)	0	L11
L10	5745727.pn.	1	L10
L9	L8 and (micro$ or macro$)	1	L9
L8	6647301.pn.	1	L8
L7	L1 and (micro$ or macro$)	1	L7
L6	L1 and updat$	1	L6

L5	L1 and track$	1	L5
L4	L1 and search$	1	L4
L3	L1 and class$	1	L3
L2	L1 and database and ind$	1	L2
L1	6453356.pn.	1	L1

END OF SEARCH HISTORY

L16: Entry 1 of 1 File: USPT Apr 28, 1998

DOCUMENT-IDENTIFIER: US 5745727 A
TITLE: Linked caches memory for storing units of information

Detailed Description Text (30):
A number of embodiments of the present invention have been described. Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the invention. For example, the linked caches of the present invention may be used in any system in which a cache is used to store first units of information associated with a second unit of information. For example, in a database in which addresses are stored in a cache, the street address may be stored in a first cache with an index into a second cache which stores the city, state, and zip code associated with the address. Accordingly, any device may be coupled to the linked caches and the coordination controller 110 to request that information be read from the linked caches.

L15: Entry 1 of 1 File: USPT Apr 28, 1998

DOCUMENT-IDENTIFIER: US 5745727 A
TITLE: Linked caches memory for storing units of information

Brief Summary Text (13):
If a requesting device requests a block of information, a first cache controller searches the first cache to determine whether the Exchange Context is present in the first cache. If the Exchange Context is not present in the first cache, then the first cache controller informs a coordination control logic device to request that a microcontroller read the Exchange Context from a main context memory array (i.e., a "context array"). The Exchange Context information read from the context array is stored in the first cache. In accordance with the present invention, the Port Context Index in the first cache is used to direct the second cache controller to associated Port Context information within the second cache. That is, the Port Context Index is communicated from the first cache to the second cache controller. The second cache controller then attempts to locate the Port Context information associated with the Exchange Context retrieved from the first cache. If the Port Context information is found, then both the Port Context information and the Exchange Context information are presented to the requesting device.

Detailed Description Text (7):
In accordance with one embodiment of the present invention, the first and second cache controllers 105, 111 are implemented as a single state machine. Operations of the first and second cache 105, 111 are preferably sequential. That is, only after the first cache controller 105 finds a requested Exchange Context does the second cache controller 111 begin searching for the associated Port Context. Preferably, the coordination controller 110 is implemented as a second state machine. A Write Path Controller 115 is preferably implemented as a third state machine. Accordingly, the cache controllers 105, 111, the coordination controller 110, and the Write Path Controller 115 each are preferably independent devices within the communications adapter 100.

CLAIMS:

1. A linked cache memory for storing units of information, the units of information being a first and second subset of units of information stored in a related memory device, including:

(a) a first cache device for storing the first subset of the information;

(b) a second cache device for storing the second subset of the information;

(c) a cache controller, coupled to the first and second cache devices for:

(1) receiving from an external device a first index;

(2) searching the first cache device for a first unit of information associated with the first index;

(3) outputting a first indication that the first unit of information was found, if the first unit of information is presently stored within the first cache device;

(4) receiving from the first cache device a second index embedded within the first unit of information;

(5) searching the second cache device for a second unit of information associated with the second index; and

(6) outputting a second indication that the second unit of information was found, if the second unit of information is present within the second cache device;

(d) a coordination controller, coupled to the cache controller, for receiving from the cache controller the first and second indications that the first and second cache devices have found the first and second units of information, and in response to such receipt of such indications, enabling outputting of the first and second units of information; and

(e) a write path controller, coupled to the coordination controller and the first and second cache devices, for coupling input signals from one of a plurality of sources to inputs of the first or the second cache, and for indicating to the source of the input signal that input data represented by the input signal has been stored in a cache device.

4. A linked cache memory for storing units of information, the units of information being a first and second subset of units of information stored in a related memory device, including:

(a) a first cache device for storing the first subset of the information;

(b) a second cache device for storing the second subset of the information;

(c) a cache controller, coupled to the first and second cache devices for:

(1) receiving from an external device a first index;

(2) searching the first cache device for a first unit of information associated with the first index;

(3) outputting a first indication that the first unit of information was found, if the first unit of information is presently stored within the first cache device;

(4) receiving from the first cache device a second index embedded within the first unit of information;

(5) searching the second cache device for a second unit of information associated with the second index; and

(6) outputting a second indication that the second unit of information was found, if the second unit of information is present within the second cache device;

(d) a coordination controller, coupled to the cache controller, for receiving from the cache controller the first and second indications that the first and second cache devices have found the first and second units of information, and in response to such receipt of such indications, enabling outputting of the first and second units of information, wherein the coordination controller is capable of performing a direct memory access operation to read information from the related memory device.

5. A linked cache memory for storing units of information, the units of information

being a first and second subset of units of information stored in a related memory device, including:

(a) a first cache device for storing the first subset of the information;

(b) a second cache device for storing the second subset of the information;

(c) a cache controller, coupled to the first and second cache devices for:

(1) receiving from an external device a first index;

(2) searching the first cache device for a first unit of information associated with the first index;

(3) outputting a first indication that the first unit of information was found, if the first unit of information is presently stored within the first cache device;

(4) receiving from the first cache device a second index embedded within the first unit of information;

(5) searching the second cache device for a second unit of information associated with the second index; and

(6) outputting a second indication that the second unit of information was found, if the second unit of information is present within the second cache device;

(d) a coordination controller, coupled to the cache controller, for receiving from the cache controller the first and second indications that the first and second cache devices have found the first and second units of information, and in response to such receipt of such indications, enabling outputting of the first and second units of information; and

(e) a lock means for locking the first and second cache devices to prevent a particular unit of information from being altered when that unit of information is in use;

wherein the first unit of information is an exchange context and the second unit of information is a port context, the exchange context and the port context being associated with a frame of data which is either being transmitted or received, and wherein the lock means activates a receive lock bit when a port context or exchange context is being used in association with a received frame.

6. A linked cache memory for storing units of information, the units of information being a first and second subset of units of information stored in a related memory device, including:

(a) a first cache device for storing the first subset of the information;

(b) a second cache device for storing the second subset of the information;

(c) a cache controller, coupled to the first and second cache devices for:

(1) receiving from an external device a first index;

(2) searching the first cache device for a first unit of information associated with the first index;

(3) outputting a first indication that the first unit of information was found, if the first unit of information is presently stored within the first cache device;

(4) receiving from the first cache device a second index embedded within the first unit of information;

(5) searching the second cache device for a second unit of information associated with the second index; and

(6) outputting a second indication that the second unit of information was found, if the second unit of information is present within the second cache device;

(d) a coordination controller, coupled to the cache controller, for receiving from the cache controller the first and second indications that the first and second cache devices have found the first and second units of information, and in response to such receipt of such indications, enabling outputting of the first and second units of information; and

(e) a lock means for locking the first and second cache devices to prevent a particular unit of information from being altered when that unit of information is in use;

wherein the first unit of information is an exchange context and the second unit of information is a port context, the exchange context and the port context being associated with a frame of data which is either being transmitted or received, and wherein the lock means activates a transmit lock bit when a port context or exchange context is being used in association with a frame to be transmitted.

7. A communications adapter within a host, for receiving and transmitting frames of data, including:

(a) a memory device for storing context data including exchange context information and port context information;

(b) a first cache device having shorter read times than the memory device, for storing a subset of the exchange context information;

(c) a second cache device having shorter read times than the memory device, for storing a subset of the port context information;

(d) a cache controller, coupled to the first and second cache device for:

(1) receiving from an external device a first index;

(2) searching the first cache device for an exchange context associated with the first index;

(3) outputting a first indication that the exchange context was found, if the exchange context being associated with the first index is presently stored within the first cache device;

(4) receiving from the first cache device a second index embedded within the exchange context;

(5) searching the second cache device for a port context associated with the second index; and

(6) outputting a second indication that the port context associated with the second index was found, if the port context is present within the second cache device;

(e) a coordination controller, coupled to the cache controller, for receiving from the cache controller the first and second indications that the first and second cache devices have found the exchange context and port context associated with the

first and second index, and enabling outputting of the exchange context and port context in response to receiving both the first and second indications; and

(f) a microcontroller, coupled to the memory device and to the coordination controller, for generating exchange context and port context information to be stored within the memory device upon receipt of a request from the coordination controller, and for directly writing to the first and second cache device.

11. A method for storing and retrieving units of information in a linked cache device having a first memory device and a second memory device, and a cache controller, the first and second memory devices each having units of information stored within that are a first and second subset, respectively, of units of information stored in a related external memory device, including the steps of:

(a) receiving from an external control device a first index;

(b) searching the first memory device for a first unit of information associated with the first index;

(c) communicating a first indication that the first unit of information was found, if the first unit of information is presently stored within the first cache device;

(d) receiving from the first memory device a second index embedded within the first unit of information;

(e) searching the second cache device for a second unit of information associated with the second index;

(f) outputting a second indication that the second unit of information was found, if the second unit of information is present within the second cache device;

(g) if the first unit of information associated with the first index is not present in the first cache device, then:

(1) performing a direct memory access operation into the related external memory device to read the first unit of information associated with the first index and store that first unit of information in the first cache device;

(2) communicating a first indication that the first unit of information associated with the first cache device is presently within the first cache device;

(h) if the second unit of information associated with the second index is not present in the second cache device, then:

(1) performing a direct memory access operation into the related external memory device to read the second unit of information associated with the second index and store that second unit of information in the second cache device; and

(2) communicating a second indication that the second unit of information associated with the second cache device is presently within the second cache device.

12. A method for storing and retrieving units of information in a linked cache device having a first memory device and a second memory device, and a cache controller, the first and second memory devices each having units of information stored within that are a first and second subset, respectively, of units of information stored in a related external memory device, including the steps of:

(a) receiving from an external control device a first index;

(b) searching the first memory device for a first unit of information associated with the first index;

(c) communicating a first indication that the first unit of information was found, if the first unit of information is presently stored within the first cache device;

(d) receiving from the first memory device a second index embedded within the first unit of information;

(e) searching the second cache device for a second unit of information associated with the second index;

(f) outputting a second indication that the second unit of information was found, if the second unit of information is present within the second cache device;

(g) receiving from the cache controller the first and second indications that the first and second cache devices have found the first and second unit of information;

(h) enabling outputting of the first and second unit of information in response to receiving both the first and second indications; and

(i) performing a direct memory access operation to read information from the external memory device.

13. A method for storing and retrieving units of information in a linked cache device having a first memory device and a second memory device, and a cache controller, the first and second memory devices each having units of information stored within that are a first and second subset, respectively, of units of information stored in a related external memory device, including the steps of:

(a) receiving from an external control device a first index;

(b) searching the first memory device for a first unit of information associated with the first index, the first unit of information being an exchange context;

(c) communicating a first indication that the first unit of information was found, if the first unit of information is presently stored within the first cache device;

(d) receiving from the first memory device a second index embedded within the first unit of information;

(e) searching the second cache device for a second unit of information associated with the second index, the second unit of information being a port context;

(f) outputting a second indication that the second unit of information was found, if the second unit of information is present within the second cache device;

(g) receiving from the cache controller the first and second indications that the first and second cache devices have found the first and second unit of information;

(h) enabling outputting of the first and second unit of information in response to receiving both the first and second indications; and

(i) locking the first and second memory devices to prevent a particular unit of information from being altered when that unit of information is in use;

(j) associating the exchange context and the port context with a frame of data which is either being transmitted or received; and

(k) activating a receive lock bit when the port context or the exchange context is being used in association with a received frame.

14. A method for storing and retrieving units of information in a linked cache device having a first memory device and a second memory device, and a cache controller, the first and second memory devices each having units of information stored within that are a first and second subset, respectively, of units of information stored in a related external memory device, including the steps of:

(a) receiving from an external control device a first index;

(b) searching the first memory device for a first unit of information associated with the first index, the first unit of information being an exchange context;

(c) communicating a first indication that the first unit of information was found, if the first unit of information is presently stored within the first cache device;

(d) receiving from the first memory device a second index embedded within the first unit of information;

(e) searching the second cache device for a second unit of information associated with the second index, the second unit of information being a port context;

(f) outputting a second indication that the second unit of information was found, if the second unit of information is present within the second cache device;

(g) receiving from the cache controller the first and second indications that the first and second cache devices have found the first and second unit of information;

(h) enabling outputting of the first and second unit of information in response to receiving both the first and second indications; and

(i) locking the first and second memory devices to prevent a particular unit of information from being altered when that unit of information is in use;

(j) associating the exchange context and the port context with a frame of data which is either being transmitted or received; and

(k) activating a transmit lock bit when the port context or the exchange context is being used in association with a frame to be transmitted.

L9: Entry 1 of 1 File: USPT Nov 11, 2003

DOCUMENT-IDENTIFIER: US 6647301 B1
TITLE: Process control system with integrated safety control system

Detailed Description Text (23):
A Compiling Translator is computer executed software which converts and translates high-level source code language software (such as the Application Program) into Machine Operation Code by using techniques of software compiling, lexical analysis, comment removal, macro substitution, statement analysis with rules, intermediate object code generation, assembly code generation, linkage to standard subroutines in Machine Operation Code form, and/or binary machine code generation.

Detailed Description Text (31):
In introduction of the control-computer-executed logic and the use of the aforementioned SEQUENCE object in a control computer program in the preferred embodiment, a brief review of two terms related to the art and technology of chemical engineering is in order. Traditional concepts of the "unit operation" (a particular kind of physical change procedure used and effected in chemical processing--for example, filtration, evaporation, distillation, or heat transfer) and the "unit process" (a particular kind of transformation in materials via chemical reaction--for example, oxidation, hydrolysis, esterification, polymerization, or nitration) define functional activities which must execute in order to effect a particular desired incremental transformation of a "starting" material to a "product" material. These physical changes and chemical transformations usually occur through use of an apparatus which, in operation, converts starting material into product material (in both a micro and macro context); and it is useful for that portion of the apparatus involved in the execution of at least one "unit operation" or "unit process" to be referenced and organized for control purposes in the described embodiment in a least one "Process Unit" (for example, a reactor, a fractionating tower, a crystallizer, a dryer, a tank farm, a distillation unit, an extraction unit, or an evaporator). In an overall chemical manufacturing plant a number of sequential and parallel "unit operations" and "unit processes" are therefore effectively executed collectively in a respective set of "Process Units" to convert many instances of "starting materials" into "product materials" (where the "product material" from one "Process Unit" is very often the "starting material" for the next "Process Unit" when the situation is examined in a micro context). Engineering and operations personnel have traditionally referenced and managed the operation of such a plant apparatus in the context of a set of such "Process Units" in effecting operation of these sequential and parallel "unit operations" and "unit processes" where "raw" ("starting") material(s) are converted into "finished" ("product") material (s) in a macro context. In example, a team of operating technicians might be "starting up" "reactor" "Process Units" in a plant even as the team is also "doing maintenance" on a first "distillation tower" "Process Unit" and "running" a second "distillation tower" "Process Unit" disposed to operate in parallel with the first distillation tower. But it needs to also be appreciated that the "Process Units" do not stand free in a physically separated and defined manner in an apparatus such as a chemical manufacturing plant (the target use of the preferred embodiment) because the chemical manufacturing plant usually enables internal fluid movement by

providing essentially one large apparatus built of conjoined vessels, pipes, pumps, valves, instruments, and wires. The "Process Unit" definitions are therefore, in verity, logical (in both cerebral and electrical-circuit-implemented contexts) and cultural rather than specifically and clearly physically defined. In example of this point, when a pipe with a valve connects a reactor to a surge tank, the reactor, pipe, valve, and surge tank are constructed as one unified and conjoined physical apparatus; one can conveniently consider the reactor as a first "Process Unit" and the surge tank as a second "Process Unit", but the issue of which Process Unit includes the valve (and the pipe) for management and control purposes requires acceptance of the fact that a technologically artistic and cultural decision must be made since (a) there is no clear and specific physical separation between the two "Process Units" within the valve and (b) bifurcated control of the valve at a particular moment of real-time is definitely not desirable.

Detailed Description Text (52):
Turning now to consideration of FIG. 3A, a block diagram of a preferred embodiment of control computer 82 is shown as control computer 10. Control computer 10 includes a Central Processor and Control Unit ("CPU") 12. In accordance with one embodiment herein, CPU 12 is based upon the MIPROC processor from Radstone Technology PLC, a Harvard architecture processor. However, it should be appreciated that other CPU circuits or microprocessors are conditionally used, and that the principles of the present invention are not limited to any particular CPU construction or integration. It should also be appreciated that all of the circuits in computer 10 are integrated into a single microcomputer chip in the appropriate application as might be achieved in a contemplated embodiment via use of ASIC (Application Specific Integrated Circuitry) technology (in conjunction with contemplated appropriate approval by the certifying agency as an acceptable certified embodiment).

27

L5: Entry 1 of 1 File: USPT Sep 17, 2002

DOCUMENT-IDENTIFIER: US 6453356 B1
TITLE: Data exchange system and method

Brief Summary Text (15):
Process monitoring, tracing, and logging are provided to track the progress of data passing through the data exchange engine and to detect and correct processing errors. In the case of a processing anomaly, the data exchange engine effects a rollback of failed transactions to preclude the loss of data. Performance statistics may also be provided.

Detailed Description Text (39):
Also shown in FIG. 9 is a statistics monitor module 264 and an associated statistics log 276 which are used to provide monitoring and tracking of data as it moves through the data exchange system. The statistics monitor module 264 also provides historical performance information on queues and historical information on system resource usage. As will be described in greater detail hereinbelow, the statistics monitor module 264 provides a means for logging and tracing a given application. Logging reveals the state of the application at the time of an error, while tracing provides a description of all software events as they occur. The tracing information may be used for tracking the application, state, and other related operations. The tracing information may be used in conjunction with the logging information to determine the cause for an error since it provides information about the sequence of events prior to an error.

Detailed Description Text (118):
The logging and tracing utility provides for logging and tracing during execution of a component. As discussed previously, logging reveals the state of the component at the time of an error, while tracing provides a description of all software events as they occur. The tracing information may be used for tracking the component-state, and other related operations. It may be used in conjunction with the logging information to determine the cause of an error, as it provides information about the sequence of events prior to an error.

CLAIMS:

7. The method of claim 1, further comprising tracking each of the data streams during converting, identifying, or transforming operations.

17. The method of claim 10, further comprising tracking the data during converting, identifying, or transforming operations.

25. The method of claim 19, further comprising: tracking processing of the information having the generic format; and logging errors occurring during the processing of the information having the generic format.

49. The medium of claim 44, further comprising tracking the data during converting, identifying, or transforming operations.

56. The system of claim 51, further comprising means for tracking the data during converting, identifying, or transforming operations.

L4: Entry 1 of 1 File: USPT Sep 17, 2002

DOCUMENT-IDENTIFIER: US 6453356 B1
TITLE: Data exchange system and method

Detailed Description Text (51):
The Common Base Class is an abstract base from which the Common Object Class is derived. An inheritance tree graphically depicting the Common Base Class is shown in FIG. 17. The main purpose of this class is to provide a single object naming and typing mechanism to aid in object tree searches and traversals. The Common Base Class is characterized in the following code-level example.

Detailed Description Text (61):
The following Common Object retrieval methods are used internally by the GetAttributeValue( ) and SetAttributeValue( ) methods to search the attribute list of a Common Object and to locate a specific Common Attribute instance. These retrieval methods may be used by application developers as well. Each of these methods require a fully dot(.) delimited Distinguished Name and will recursively walk all relative levels of nesting to retrieve the relevant object/attribute.

Detailed Description Text (221):
Dequeue( ) attempts to find a queue object marked as NORMAL_OBJECT first from the memory buffer. If it can not find one, it will try to find one from the queue files. If Dequeue( ) finds a queue object marked as NORMAL_OBJECT in a file, it creates a queue object, marks it as ACTIVE_OBJECT, inserts it into the memory buffer, and de-serializes the Common Object it refers to and returns the Common Object to the caller. During this process, if Dequeue( ) can not de-serialize the Common Object, it will mark the queue object as DELETED_OBJECT in the queue file and continue its search.

DOCUMENT-IDENTIFIER: US 6453356 B1
TITLE: Data exchange system and method

Detailed Description Text (113):
Two macros, DX_SYSINIT and DX_SYSEXIT, are used to manage initialization and destruction of the DX_SysConfigObject, respectively. A usage example of these two macros is given as follows:

Detailed Description Text (125):
In order to write a message into the log/trace file, the developer may use the macro DX_TL as shown below: DX_TL(DX_ARGS,Category, StringToLog/ErrorNumber[,arg 1 [,arg2]]);

Detailed Description Text (126):
The macro DX_ARGS includes parameters such as filename, line number, time and thread ID that are automatically written into the trace/log messages. Category is specified by the following enumerated data types:

Detailed Description Text (134):
An error/event may occur at a very low level in the code (e.g., database space exhausted). It is important to report this low level event, but it is also important to report the context of what was trying to be achieved within the application when this low level error occurred. The application developer is provided with macros to define a context within the developer's code. The set of macros provided for this purpose include: INIT_CONTEXT; CONTEXT_BEGIN; and CONTEXT_END. In general, every function using the context macros should first use the macro INIT_CONTEXT. It is noted that, if INIT_CONTEXT is not called before defining CONTEXT_BEGIN, the code may not compile.

Detailed Description Text (135):
The beginning of a context may be defined using the macro CONTEXT_BEGIN, and the end of a context can be defined using the macro CONTEXT_END, as is indicated in the following example. The CONTEXT_BEGIN macro takes the argument Context Number. This context number is used to access the Context Catalog of an application and to retrieve the context string. It is noted that nested contexts are generally not allowed. If a CONTEXT_BEGIN is called before the previous context is ended, an implicit CONTEXT_END for the previous context is assumed. The following example is provided:

Detailed Description Text (140):
Within a given function, INIT_CONTEXT declares a pointer to a DX_ContextObject, referred to as dx_context, and initializes it to point to a global dummy DX_ContextObject, whose context string is blank. It also declares and initializes a variable dx_init_context. The definition of the INIT_CONTEXT macro is as follows:

Detailed Description Text (142):
The macro CONTEXT_BEGIN, described in the following example, checks whether dx_init_context is initialized or not. The significance of this check is to make

sure that the function does not compile if INIT_CONTEXT is not called before the first occurrence of CONTEXT_BEGIN. It then initializes the DX_ContextObject pointer to point to a new DX_ContextObject instance storing the context string specified by the context number argument.

Detailed Description Text (144):
The macro CONTEXT_END deletes the DX_ContextObject instance created by CONTEXT_BEGIN, as can be seen in the following example.

Detailed Description Text (169):
As in the case of run-time configuration management, the ReconfigParameters( ) function on DX_SysConfigObject will be called. In this function, the DX_SysConfigObject first checks if the signal/event received corresponds to Shutdown and if the PID specified is its own PID. If so, it, in turn, must make sure that no new transactions are started, and waits for all of the current transactions to be completed. This involves calling the macro DX_SYSEXIT. It is noted that, before shutting down, the entry in the configuration file should be deleted by the exiting process. It is possible that the component aborts prior to cleaning up the configuration file. This stray entry does not effect the start up of any other component using the same configuration file. DX_ConfigSet is also responsible for clean up of stray DX_SHUTDOWN entries in the configuration file.

Detailed Description Text (181):
A macro called DX_Thread_Execute( ) is provided for ease of use. This macro retrieves the DX_ThreadController instance from the DX_SysConfigObject and then invokes the DX_ThreadController::Execute( ) method. The method DX_ThreadController::Execute( ) behaves exactly the same as if a call was invoked to create a new thread. A pointer must be passed to the function and as well as a pointer to the arguments. Internally, the DX_ThreadController uses the class DX_ThreadRequest when a thread is not available to provide a FIFO buffer that will store the function pointer and argument pointer. Each time a thread completes execution, the FIFO is checked for the presence of entries. If there are entries in the FIFO, the first entry in the buffer is removed and executed. An example of DX_ThreadController implementation is provided in the following example:

DOCUMENT-IDENTIFIER: US 6453356 B1
TITLE: Data exchange system and method

Drawing Description Text (14):
FIG. 17 is an illustration of an inheritance tree graphically depicting a Common Base Class associated with the Common Object shown in FIG. 15;

Drawing Description Text (15):
FIG. 18 is a class structure diagram showing public and non-public interfaces associated with various file based and database based queuing processes;

Drawing Description Text (16):
FIG. 19 is a pictorial description of the calling structure between data exchange queue classes when dequeueing a Common Object in accordance with one embodiment of the present invention; and

Drawing Description Text (17):
FIG. 20 is a pictorial description of the calling structure between data exchange queue classes when enqueueing a Common Object in accordance with the embodiment of FIG. 19.

Detailed Description Text (46):
A Common Object within the context of this illustrative embodiment represents a C++ container object that is used to contain multiple portions of attribute data within a single flexible object. A Common Object comprises two lists, a Private List and a Public List. Both lists contain one or more attribute/value pairs (AV pairs). These AV pairs represent objects referred to as Common Attribute Objects, each of which comprises an attribute name, value, and type. Common Attribute classes are available for all of the basic types plus some complex types as well.

Detailed Description Text (47):
The Public List represents a sequence of two types of user-defined attributes, which are instances of an Attribute class or a Common Object. The Private List contains attributes that are used internally by the system for a variety of purposes, such as naming, routing, and identifying ancestry information. The Public List contains data that the user has defined, which may include other Common Objects. Each list may contain two types of attributes, type specific AV pairs or objects. Contained objects may either be List Objects or other Common Objects. The Private List does not include contained objects since this List is only used for simple tags or header information.

Detailed Description Text (51):
The Common Base Class is an abstract base from which the Common Object Class is derived. An inheritance tree graphically depicting the Common Base Class is shown in FIG. 17. The main purpose of this class is to provide a single object naming and typing mechanism to aid in object tree searches and traversals. The Common Base Class is characterized in the following code-level example.

Detailed Description Text (79):
The use of reference counting greatly reduces the amount of time and memory that is required to copy objects. Use of a third-party foundation class library, such as one developed by Rogue Wave Software, Inc., automatically supplies a number of the copy constructors. Also, methods within the DX_CommonObject class itself make use of the Rogue Wave copy methods as well. It is noted that the DX_StringAttribute and DX_BlobAttribute classes provide their own copy optimization through reference counting, as objects of these classes could be of a substantial size.

Detailed Description Text (80):
The Common Attribute is an object that is contained within a Common Object and is used to contain attribute data. The Common Attribute contains a private attribute that denotes the specific attribute type and a set of public attributes for name and value. A Common Attribute may be a simple attribute of a specific data type, name, and value or it may contain another object, such as a List Object or Common Object. The type specific Common Attribute classes all inherit their capabilities from the generic Common Attribute class so all classes will behave in an equivalent manner. Reference is made to FIGS. 16A-16D, which illustrate the contents of a Common Attribute when used to represent some of the supported data types.

Detailed Description Text (81):
The Common Attribute class is an abstract base class from which type specific Attribute classes are derived, a characterizing example of which is given as follows:

Detailed Description Text (88):
Code-level examples of various type specific attribute classes that are supported are provided as follows:

Detailed Description Text (91):
EXAMPLE #20 class: DX_String The DX_String class is a reference counted container class used by the DX_StringAttribute to store the attribute's value. It provides the user a way to keep down the overhead associated with having to copy the data.

Detailed Description Text (103):
As was discussed previously, a List is a sequence of attributes, i.e., instances of attribute classes and/or Common Objects, an example of which is given as follows:

Detailed Description Text (112):
When initiated, a component creates an instance of a System Configuration Object (DX_SysConfigObject) that stores the current parameter settings. The component also registers for a Signal/Event so that it is informed of changes to the configuration using the dynamic configuration command line interface/web interface. When a user wants to change the run time parameters of a component (identified by the process ID and the machine on which it is running), a signal/event is sent to the component to update its configuration. A signal/event handler invokes the ReconfigParameters ( ) method on the DX_SysConfigObject, which takes care of reconfiguring the various controller objects, such as DX_TraceLogObject, DX_QueueManager, and DX_ThreadController for example. The System Configuration object, DX_SysConfigObject, is a singleton object that initializes and controls the configuration of a component in the data exchange system, such as logging/tracing levels, thread controller, queuing, and performance monitoring. A singleton object, as understood in the art, refers to a C++ nomenclature meaning that only a single instance of the class may exist within a single executable. A singleton object is most commonly used when controlling system wide resources.

Detailed Description Text (114):
EXAMPLE #34 #define DX_SYSINIT(ComponentName) .backslash.
DX_SysConfigObject::Instance(ComponentName); .backslash. RegisterForEvent( );
#define DX_SYSEXIT EndWaitForEvent( ); .backslash.

DX_SysConfigObject::DeleteInstance( ); class DX_SysConfigObject{ public: // To ensure that only one instance of the System Config object // exists, one has to always use this function to obtain a // reference to the // system config object // Cannot delete the pointer returned. use // DX_SYSEXIT if you want to delete // the sysconfigobject static DX_SysConfigObject* Instance(char *componentName=0); // static method to delete the instance of the singleton object static void DeleteInstance( ); // Called when the parameters are to be changed to run time void ReconfigParameters( ); // used to find parameter values by name from the paramValueList // do not delete the pointer returned by this function char *FindValue(char *name); private: // constructor is private to make sure the user cannot instantiate DX_SysConfigObject(char *componentName); // destructor .about.DX_SysConfigObject( ); // to read the configuration file and initialize the // paramValueList EreturnCodes InitParamValueList(char *dx_home, char** outCfgFileName); void GetTraceLogParams(DX_INDICATOR *trcLogCategoryInd, char* trcLogDir, long &logSize); // a pointer that stores the one and only instance of the // system config object static DX_SysConfigObject* instance; // the list of various configuration parameters //DX_ListObject *paramValueList; RWDlistCollectables *paramValueList; char ComponentName[MAX_NAME_LEN]; // Pointer to DX_ThreadController instance DX_ThreadController *PthrCtrl; // Method to instantiate DX_ThreadController // after DX_SysConfigObject constructor EreturnCodes InitThreadController( ); void DeleteThreadController( ); void DeletePMonitor( ); // Pointer to DX_Monitor instance DX_Monitor *pMonitor; // Method to instantiate DX_Monitor after // DX_SysConfigObject constructor EreturnCodes InitMonitor(const char *appName); EreturnCodes InitTraceLogObject (char* componentName); // Get Monitor config parameters void GetMonitorParam(struct MonConfigType *monConfig); // To access the DX_ThreadController object DX_ThreadController* GetThreadController( ); DX_Mutex* paramListLock; };

Detailed Description Text (157):
The Monitor Object is instantiated by the DX_SysConfigObject instance or by calling the static method DX_Monitor::Instance( ) directly. There is only one DX_Monitor thread running per executable component. The monitor thread is spawned whenever the MONITOR in the system configuration file is triggered to ON. The monitor thread exists until the MONITOR is triggered to OFF. The implementation of the monitor impacts three areas. The Queue Manager provides the queue performance data. The DX_SysConfigObject provides the configuration change handling and the monitor object instantiation. The DX_Monitor Object spawns or kills the monitor and generates the monitor log files or log table in the database. The methods added in the DX_Queue classes are listed below:

Detailed Description Text (161):
An additional method added in the DX-Queue classes is provided as follows:

Detailed Description Text (175):
The queue viewing administration and maintenance utility will now be described. DX_Qview <queue_name>[priority] permits viewing of all items in a Queue identified by its name. This information may be obtained using the GetQueueView( ) method on DX_QueueManager object. DX_GetCO <oidval> permits viewing of the common object for a particular OID in the queue. This utility uses the Demarshal( ) method provided in DX_CommonObject class. Other viewing options include the following: viewing the queue entry corresponding to common object specified by its name or OID; viewing all queue entries enqueued by a particular source; viewing all queue entries having a particular status; and viewing the names of all objects in the queue.

Detailed Description Text (181):
A macro called DX_Thread_Execute( ) is provided for ease of use. This macro retrieves the DX_ThreadController instance from the DX_SysConfigObject and then invokes the DX_ThreadController::Execute( ) method. The method DX_ThreadController::Execute( ) behaves exactly the same as if a call was invoked to create a new thread. A pointer must be passed to the function and as well as a

pointer to the arguments. Internally, the DX_ThreadController uses the class DX_ThreadRequest when a thread is not available to provide a FIFO buffer that will store the function pointer and argument pointer. Each time a thread completes execution, the FIFO is checked for the presence of entries. If there are entries in the FIFO, the first entry in the buffer is removed and executed. An example of DX_ThreadController implementation is provided in the following example:

Detailed Description Text (183):
A DX_Utils library provides the DX_Mutex class for platform independent mutex protection, an example of which is provided below. The DX_Mutex class does not require use of the DX_ThreadController class.

Detailed Description Text (185):
For purposes of data internationalization, Unicode UTF-8 formatting is provided to store all attribute value strings using wide character strings. Code conversion functions to convert a Unicode string to UTF-8 string and vice-versa are also provided. These conversion methods are used to store any user-specified data internally in UTF-8 format. To support language localization, all message strings use an external message catalog. The interface provided by the DX_CodeConversion class is as follows:

Detailed Description Text (197):
As was discussed previously, two types of priority based queues are used, namely, the incoming Receive Queues and the outgoing Send Queues. Each outgoing adapter will have its own outgoing queue so that any interface specific translation or routing may be performed outside the engine core. Each instance of the DX_Engine executable has one or more input queues, although only one is allowed for file-based queues, and one or more output queues. An instance of the DX_QueueManager class is used as a central proxy to all queue access, and will be mutex protected and record-lock protected, for file-based implementation, or row lock protected, for database implementations, to prevent data contention.

Detailed Description Text (202):
The Queue Manager public interface makes use of the DX_QueueTransaction object for transaction control. The Enqueue( ), Dequeue( ), Commit( ), and Rollback( ) methods take pass-in argument of an instance of the DX_QueueTransaction class which belongs to a running thread. The transaction object contains an ordered list of operations performed in this transaction. For file-based implementations, all operations are maintained in buffered memory and are not written into file storage until commit time. For database implementations, the database provided rollback mechanism is employed, with each transaction using its own unique run-time database context.

Detailed Description Text (203):
The class structure diagram is shown in FIG. 18. The public interface, shown in shaded boxes, is used by adapter developers, but the non-public interface, shown without shading, is intended for internal use only. This usage restriction is forcefully implemented.

Detailed Description Text (206):
The DX_QueueManager class is a singleton class and acts as the global access point of all queue operations. It contains a list of queues, instances of DX_Queue class, as its data member. Users, however, do not need to create a queue with Queue Manager before using it. The creation of the queue is embedded in Enqueue and Dequeue operations. Besides the Enqueue ( ) and Dequeue ( ) operations, DX_QueueManager also defines interfaces for extended transaction support, performance monitoring, and queue administration. An illustrative example of DX_QueueManager implementation is provided as follows:

Detailed Description Text (210):
DX_Queue is an abstract interface class. It only provides an interface for

Enqueue/Dequeue operations and Commit/Rollback operations. An implementation example is given as follows:

Detailed Description Text (212):
DX_FileQueue class contains four child queues for each priority. Besides the queue operation interface and transaction interface, the algorithm of priority handling is also implemented in this class. The priority algorithm is implemented inside the Dequeue method. The DX_IndexObject argument of the Dequeue methods is used for transaction control. At running time, Dequeue operations fill in corresponding fields in DX_IndexObject, which is a component of DX_QueueOperation object. An implementation example is given as follows:

Detailed Description Text (217):
A logical dequeue cursor is also defined and manipulated in this class. For purposes of performance, this cursor should never be rolled back. This feature is also implemented via the help of DX_QueueObjectList class as well.

Detailed Description Text (225):
The DX_IndexObject class is important for all the queue operations. It is used to uniquely identify the location at which a Common Object is stored, and is further used for internal routing of all queue requests. The DX_IndexObject class is member of the DX_QueueObject class and DX_QueueOperation class. Its members include queue name, queue priority, storage type, file handle, file index, record offset, and record sequence. An illustrative example of a DX_IndexObject implementation is provided as follows:

Detailed Description Text (232):
Objects are stored and retrieved from a persistent store in an efficient manner. Two types of object storage, termed relational database storage and file storage, are implemented for this purpose. The object persistency is implemented using Marshal( ), Demarshal( ), and DeleteStorage( ) methods of the DX_CommonObject class, where a parameter is passed to indicate storage type. A policy for object caching and sharing may also be employed.

Detailed Description Text (237):
It is noted that when serializing and de-serializing an object in a Java.TM. environment, the Rouge Wave Jtools Java.TM. foundation classes may be used since they project a broad set of base classes and are compatible with the Rogue Wave Tools.h++ classes used in C++.

Detailed Description Paragraph Table (1):
class: DX_CommonBase : public RWCollectable { RWDECLARE_COLLECTABLE(DX_CommonBase); protected: char* Name; int Type; void SetName(const char* inName); public: char* GetName() { return name; } int GetType() { return type; } virtual void PrintContents()=0; char* GetTypeName(int type); };

Detailed Description Paragraph Table (2):
enum EclassTypes { DX_OBJECT = 0, DX_ATTRIBUTE, DX_LIST, DX_MULTIVALUE, DX_INTEGER, DX_STRING, DX_STRINGVAL, DX_DATE, DX_TIME, DX_REAL, DX_BLOB, DX_BLOBVALUE, DX_VERSION, DX_OID_CLASS }; enum EreturnCodes { NOT_FOUND = 0, SUCCESS, FAILED, TIME_OUT, ACCESS_DENIED, DUPLICATE_OBJECT, NO_OBJECT, NO_ATTRIBUTE, INVALID_ATTRVAL, INVALID_ARGS, INVALID_OPERATION, SYSTEM_ERROR }; enum EstorageTypes { FLATFILE = 0, DATABASE }; enum ElistType { PUBLIC = 0, PRIVATE }; static const int MAX_NAME_LEN=255; static const int MAX_DOTTED_NAMELEN=8096; static const int MAX_MULTI_VAL=255; static const int MAX_BLOB_SIZE=2048; static const int OID_LEN=128; static const int MAX_FILE_NAME=256; static const int MAX_PID_LEN=12; static const int MAX_LINE_SIZE=100;

Detailed Description Paragraph Table (3):
class: DX_CommonObject: public DX_CommonBase { RWDECLARE_COLLECTABLE

(DX_CommonObject); friend class DX_ListObject; friend class DX_FileSubQueue;
public: virtual.about.DX_CommonObject(); //
Constructors /***/ //I
name==NULL, the name is set to "NOT_SET". //If name is ""or contains a "." it will
be set to
"INVALID_NAME" /***/
DX_CommonObject(const char*
name=0); /***/ //Creat
a copy of an entire DX_CommonObject, //but with it's own unique OID
value /***/
DX_CommonObject* Clone
(); /***/ //All
AddAttribute and AddPvtAttribute methods take ownership of //the pointers passed in
to them. Do NOT delete the pointers after //a call to AddAttribute and
AddPvtAttribute. The pointers will be //deleted when the container DX_CommonObject
or DX_ListObject is deleted. // //NOTE: When using the following two methods for
creating a DX_STRING attribute // // AddAttribute(const char* name, const int type,
const void* value) // and AddPvtAttribute(const char* name, const int type, const
void* value) // // the value is defaulted to be of type const char* and not
UNICHAR* // Misuse will lead to unexpected
behavior. /***/
EreturnCodes AddAttribute(DX_CommonAttribute* newAttr); EreturnCodes AddAttribute
(DX_ListObject* newAttr); EreturnCodes AddAttribute(DX_CommonObject* newAttr);
EreturnCodes AddAttribute(const char* name, const int type, const void* value);
EreturnCodes AddPvtAttribute(DX_CommonAttribute* newAttr); EreturnCodes
AddPvtAttribute(DX_ListObject* newAttr); EreturnCodes AddPvtAttribute
(DX_CommonObject* newAttr); EreturnCodes AddPvtAttribute(const char* name, const
int type, const void*
value); /***/ //Delete
and DeletePvtAttribute will remove the named attribute //from the container
DX_CommonObject or DX_ListObject and delete the named //attribute's
pointer /***/
EreturnCodes DeleteAttribute(const char* name); EreturnCodes DeletePvtAttribute
(const char*
name); /***/ //RemoveA
will remove the named attribute from the container //DX_CommonObject or
DX_ListObject but will not delete the named //attribute's
pointer /***/
EreturnCodes RemoveAttribute(const char* name); EreturnCodes RemovePvtAttribute
(const char*
name); /***/ //Do NOT
delete the pointer that is returned to you, it still is owned by //the container
DX_CommonObject or DX_ListObject. You CAN use the any of //the attribute class
methods for the
pointer. /***/ void*
GetAttribute(const char* name); void* GetPvtAttribute(const char* name); void
PrintContents(); //The caller of Demarshal is responsible for object's memory
allocation static DX_CommonObject* Demarshal(char* ObjOid, int type, int
ContextIndex); EreturnCodes Marshal(int type, int ContextIndex); static
EreturnCodes DeleteStorage(const char* oidVal, int type, int ContextIndex); //The
caller of GetOID is responsible for deleting the memory of the returned char* char*
GetOID(); EPriorityCode GetPriority(); protected: void* Find(const char* name);
static EreturnCodes RestorePersistentObject(const char* oidVal); static
EreturnCodes DeletePersistentObject(const char* oidVal, int type); private: //Data
Members DX_ListObject* PrivateList; DX_ListObject* PublicList; static void Delete
(RWDlistCollectables* list); static int Copy(RWDlistCollectables* dest, const
RWDlistCollectables *src); static EreturnCodes CopyPersistentObject(char*
filename); static EreturnCodes CopyFile(char* filename,char* copyfilename); void
saveGuts(RWFile& file) const; void saveGuts(RWvostream& stream) const; void
restoreGuts(RWFile& file); void restoreGuts(RWvistream& stream); const int check()

const; DX_Boolean updateListOids(DX_ListObject* Plist, DX_OID *Poid); DX_Boolean updateObjectOids(DX_CommonObject* Pobj, DX_OID *Poid); DX_Boolean ValidateNaming (const char* name); #ifdefDX_DATABASE EreturnCodes InsertIntoTable(char* filename, int ContextIndex); static EreturnCodes RetrieveFromTable(char* filename, char* oid, int ContextIndex); EreturnCodes UpdateTable(char* filename, int ContextIndex); #endif //use to cal. the number of bytes needed to store an object using RWFile RWspace binaryStoreSize() const;}; };

Detailed Description Paragraph Table (5):
class: DX_OID: public DX_ComrnonAttribute { RWDECLARE_COLLECTABLE(DX_OID); friend class DX_SysConfigObject; friend class DX_CommonObject; friend class DX_ListObject; protected: DX_OID(char* value); DX_OID(const char* name, char* value); public: // constructors DX_OID(); virtual.about.DX_OID
(); /***/ // Get a copy of type specific value. User should new a pointer for the // return value and then delete the pointer after
use /***/
EreturnCodes GetAttributeValue(void* value); EreturnCodes GetAttributeValue(char* value); void PrintContents(); private: static unsigned long OIDCOUNT; #ifndef HPUX static DX_Mutex lockOidCount; #endif char *AttrValue; char *storeFileName; char *storeDirName; void saveGuts(RWFile& file) const; void saveGuts(RWvostream& stream) const; void restoreGuts(RWFile& file); void restoreGuts(RWvistream& stream); EreturnCodes SetAttributeValue(void* value); char* GetDirName(); char* GetFileName (); static DX_Boolean CheckOIDFormat(char* oidVal); };

Detailed Description Paragraph Table (6):
class: DX_CommonAttribute : public DX_CommonBase { public: virtual EreturnCodes GetAttributeValue(void* value) = 0; virtual EreturnCodes SetAttributeValue(void* value) = 0; virtual void PrintContents()=0; };

Detailed Description Paragraph Table (7):
attribute class:
DX_IntegerAttribute /***
* CLASS NAME : DX_IntegerAttribute * INHERITED FROM : None * INHERITS : DX_CommonAttribute * DESCRIPTION : Provides storage for integer attributes
***/ class
DX_IntegerAttribute:public DX_CommonAttribute { RWDECLARE_COLLECTABLE (DX_IntegerAttribute);
public: /***/ // If name==NULL, the name is set to "NOT_SET". // If name is ""or contains a "." it will be set to "INVALID
NAME" /***/
DX_IntegerAttribute(const char* name=0); DX_IntegerAttribute(const char* name, int value); virtual.about.DX_IntegerAttribute
(); /***/ // The memory for the value argument is allocated and deallocated // by the caller. The library function GetAttributeValue just writes // to the value argument and the SetAttributeValue just reads the // argument to reset the value of the attribute /***/
EreturnCodes GetAttributeValue(void* value);// argument assumed to be int*
EreturnCodes GetAttributeValue(int& value); EreturnCodes SetAttributeValue(void* value);// argument assumed to be int* EreturnCodes SetAttributeValue(int value); void PrintContents(); private: int AttrValue; void saveGuts(RWFile& file) const; void saveGuts(RWvostream& stream) const; void restoreGuts(RWFile& file); void restoreGuts(RWvistream& stream); };

Detailed Description Paragraph Table (8):
attribute class:
DX_StringAttribute /***
* CLASS NAME : DX_StringAttribute * INHERITED FROM : None * INHERITS :

DX_CommonAttribute * DESCRIPTION : Provides storage of strings.
**********/ class
DX_StringAttribute : public DX_CommonAttribute { RWDECLARE_COLLECTABLE
(DX_StringAttribute);
public: /**********/ // If
name==NULL, the name is set to "NOT_SET". // If name is "" or contains a "." it
will be set to "INVALID
NAME" /**********/
DX_StringAttribute(const char* name=0), DX_StringAttribute(const char* name, const
char* value); DX_StringAttribute(const char* name, const UNICHAR* value);
DX_StringAttribute(const char* name, const DX_String& value);
virtual.about.DX_StringAttribute
(); /**********/ // Get a copy
of char* value. The library will allocate the // appropriate storage, the user
should delete the pointer after
use. /**********/ EreturnCodes
GetAttributeValue(void* value);// assumes char** is passed // Get a copy oftype
specific value. The library will allocate the // appropriate storage, the user
should delete the pointer after
use. /**********/ EreturnCodes
GetAttributeValue(char* &value); EreturnCodes GetAttributeValue(UNICHAR* &value);
EreturnCodes GetAttributeValue(DX_String*
&value); /**********/ // The
memory for the value argument is allocated and deallocated // by the caller. The
library functions just read the value // argument to reset the value of the
attribute. // NOTE: the method taking (void* value) assumes const
char* /**********/ EreturnCodes
SetAttributeValue(void* value); EreturnCodes SetAttributeValue(const char* value);
EreturnCodes SetAttributeValue(const UNICHAR* value); EreturnCodes
SetAttributeValue(const DX_String& value); void PrintContents(); private:
DX_String* AttrValue; void saveGuts(RWFile& file) const; void saveGuts(RWvostream&
stream) const; void restoreGuts(RWFile& file); void restoreGuts(RWvistream&
stream); };

Detailed Description Paragraph Table (9):
class DX_String : public RWCollectable { RWDECLARE_COLLECTABLE(DX_String) friend
class DX_StringAttribute; public: DX_String(); DX_String(const char* value);
virtual.about.DX_String(); void PrintContents(); private: char* StringValue; int*
len; int* refCount; DX_String& operator=(const DX_String& rhs); void saveGuts
(RWFile& file) const; void saveGuts(RWvostream& stream) const; void restoreGuts
(RWFile& file); void restoreGuts(RWvistream& stream); };

Detailed Description Paragraph Table (10):
attribute class:
DX_DateAttribute /**********
* CLASS NAME : DX_DateAttribute * INHERITED FROM : None * INHERITS :
DX_CommonAttribute * DESCRIPTION : Provides storage for Date attributes * * NOTE:
This class only stores the date related fields of the struct tm* * The time related
fields are initialized to 0, any data passed in * the struct tm* time related
fields will be discarded.
**********/ class
DX_DateAttribute : public DX_CommonAttribute { RWDECLARE_COLLECTABLE
(DX_DateAttribute);
public: /**********/ // If
name==NULL, the name is set to "NOT_SET". // If name is ""or contains a "." it will
be set to
"INVALID_NAME" /**********/
DX_DateAttribute(const char* name=0); DX_DateAttribute(const char* name, const
struct tm value); virtual DX_DateAttribute
(); /**********/ // The struct

tm passed as an argument for GetAttributeValue // is to be allocated and deallocated by the caller // The library function just copies the value into the // structure. /***/ EreturnCodes GetAttributeValue(struct tm& value); EreturnCodes GetAttributeValue (void* value); // assumes struct tm* is passed /***/ // The memory for the value argument is allocated and deallocated // by the caller. The library functions just read the value // argument to reset the value of the attribute /***/ EreturnCodes SetAttributeValue(void * value); EreturnCodes SetAttributeValue(const struct tm value); void PrintContents(); private: RWCollectableDate *AttrValue; void saveGuts(RWFile& file) const; void saveGuts(RWvostream& stream) const; void restoreGuts(RWFile& file); void restoreGuts(RWvistream& stream); };

Detailed Description Paragraph Table (11):
attribute
class:DX_TimeAttribute /***
** CLASS NAME : DX_TimeAttribute * INHERITED FROM : None * INHERITS : DX_CommonAttribute * DESCRIPTION : Provides storage of time values in form of SSE. ***/ class DX_TimeAttribute : public DX_CommonAttribute { RWDECLARE_COLLECTABLE (DX_TimeAttribute); public: /***/ // If name==NULL, the name is set to "NOT_SET". // If name is ""or contains a "." it will be set to "INVALID_NAME" /***/ DX_TimeAttribute(const char*name=0); DX_TimeAttribute(const char*name, unsigned long value); virtual.about.DX_TimeAttribute (); /***/ // The memory for the value argument is allocated and deallocated // by the caller. The library function GetAttributeValue just writes // to the value argument and the SetAttributeValue just reads the // argument to reset the value of the attribute /***/ // argument assumed to be unsigned long* EreturnCodes GetAttributeValue(void* value); EreturnCodes GetAttributeValue(unsigned long& secondSinceEpoch); // argument assumed to be unsigned long* EreturnCodes SetAttributeValue(void* value); EreturnCodes SetAttributeValue(const unsigned long value); void PrintContents(); private: RWCollectableTime* AttrValue; void saveGuts(RWFile& file) const; void saveGuts(RWvostream& stream) const; void restoreGuts(RWFile& file); void restoreGuts(RWvistream& stream); };

Detailed Description Paragraph Table (12):
attribute
class:DX_RealAttribute /** *
CLASS NAME : DX_RealAttribute * INHERITED FROM : None * INHERITS : DX_CommonAttribute * DESCRIPTION : Provides storage of float types **/ class DX_RealAttribute : public DX_CommonAttribute { RWDECLARE_COLLECTABLE(DX_RealAttribute); public: /***/ //If name==NULL, the name is set to "NOT_SET". //If name is ""or contains a "." it will be set to //"INVALID_NAME" /***/ DX_RealAttribute(const char* name=0); DX_RealAttribute(const char* name, float value); virtual .about.DX_RealAttribute ( ); /***/ //The memory for the value argument is allocated and //deallocated by the caller. The library function //GetAttributeValue just writes to the value argument //and the SetAttributeValue just reads the argument //to reset the value of the attribute /***/ EreturnCodes GetAttributeValue(void* value); //argument assumed to be float* EreturnCodes GetAttributeValue(float& value); EreturnCodes SetAttributeValue(void*

value); //argument assumed to be float* EreturnCodes SetAttributeValue(float value); void PrintContents( ); private: float AttrValue; void saveGuts(RWFile& file) const; void saveGuts(RWvostream& stream) const; void restoreGuts(RWFile& file); void restoreGuts(RWvistream& stream); };

Detailed Description Paragraph Table (13):
attribute
class:DX_BlobAttribute /** * CLASS NAME : DX_BlobAttribute * INHERITED FROM : None * INHERITS : DX_CommonAttribute * DESCRIPTION : Attribute storage class to store raw * binary stream * stored in a unsigned char* * Takes/returns DX_Blob struct **/ class DX_BlobAttribute : public DX_CommonAttribute { RWDECLARE_COLLECTABLE(DX_BlobAttribute); public: /***/ //If name==NULL, the name is set to "NOT_SET". //If name is "" or contains a "." it will be set to //"INVALID_NAME" /***/ DX_BlobAttribute(const char* name=0); DX_BlobAttribute(const char* name, const DX_Blob& value); DX_BlobAttribute(const char* name, unsigned char* value, unsigned int size); virtual .about.DX_BlobAttribute ( ); /***/ //Get a copy of DX_Blob* value. The library will allocate the //appropriate storage, the user should delete the pointer after use. /***/ EreturnCodes GetAttributeValue(void* value);//assumes DX_Blob** /***/ //Get a copy of type specific value. The library will allocate the //appropriate storage, the user should delete the pointer after use. /***/ EreturnCodes GetAttributeValue(DX_Blob* &value); EreturnCodes GetAttributeValue(unsigned char* &value); /***/ //The memory for the value argument is allocated and //deallocated by the caller. The library functions just read //the value argument to reset the value of the attribute. // //NOTE: the method taking (void* value) assumes //const DX_Blob* /***/ EreturnCodes SetAttributeValue(void* value); EreturnCodes SetAttributeValue(const DX_Blob& value); EreturnCodes SetAttributeValue(const unsigned char* value, unsigned int size); unsigned int GetBlobSize( ); void PrintContents( ); private: DX_Blob* AttrValue; void saveGuts(RWFile& file) const; void saveGuts(RWvostream& stream) const; void restoreGuts(RWFile& file); void restoreGuts(RWvistream& stream); };

Detailed Description Paragraph Table (14):
class: DX_Blob The DX_Blob class is a reference counted container class used by the DX_BlobAttribute to store the attribute's value. It provides the user a way to keep down the overhead associated with having to copy the data. class DX_Blob : public RWCollectable { RWDECLARE_COLLECTABLE(DX_Blob); friend class DX_BlobAttribute; friend class DX_MultiValueAttribute; public: DX_Blob( ); DX_Blob(const unsigned char *value, unsigned int size); .about.DX_Blob( ); DX_Blob& operator=(const DX_Blob& rhs); void PrintContents( ); unsigned int GetBlobSize( ); EreturnCodes GetValue(unsigned char** value); private: int* refCount; unsigned int* BlobSize; unsigned char* BlobValue; void saveGuts(RWFile& file) const; void saveGuts (RWvostream& stream) const; void restoreGuts(RWFile& file); void restoreGuts (RWvistream& stream); };

Detailed Description Paragraph Table (15):
attribute
class:DX_VersionAttribute /** * CLASS NAME : DX_VersionAttribute * INHERITED FROM : None * INHERITS : DX_CommonAttribute * DESCRIPTION : Provides a integer value that * : can be used to mark the version of * : an object in process. **/ class DX_VersionAttribute : public DX_CommonAttribute { RWDECLARE_COLLECTABLE

L1: Entry 1 of 3 File: USPT Jan 16, 2001

US-PAT-NO: 6175830
DOCUMENT-IDENTIFIER: US 6175830 B1
**** See image for Certificate of Correction ****

TITLE: Information management, retrieval and display system and associated method

DATE-ISSUED: January 16, 2001

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Maynard; George	Wooster	OH		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
EvREsearch, Ltd.	Columbus	OH			02

APPL-NO: 09/ 315316 [PALM]
DATE FILED: May 20, 1999

INT-CL: [07] G06 F 17/30

US-CL-ISSUED: 707/5; 704/9, 706/47
US-CL-CURRENT: 707/5; 704/9, 706/47

FIELD-OF-SEARCH: 707/1, 707/2, 707/3, 707/4, 707/5, 704/9, 706/47

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	5265065	November 1993	Turtle	707/4
☐	5404510	April 1995	Smith et al.	707/2
☐	5537586	July 1996	Amram et al.	707/3
☐	5546529	August 1996	Bowers et al.	395/348
☐	5642502	June 1997	Driscoll	707/5
☐	5675710	October 1997	Lewis	706/12
☐	5708806	January 1998	DeRose et al.	707/104
☐	5778157	July 1998	Oatman et al.	706/46

h e b b g e e e f c e ef g e ge

☐	5784608	July 1998	Meske, Jr. et al.	707/2
☐	5790121	August 1998	Sklar et al.	345/356
☐	5848410	December 1998	Walls et al.	707/4
☐	5855015	December 1998	Shoham	707/5
☐	5864863	January 1999	Burrows	707/103
☐	5870559	February 1999	Leshem et al.	709/224
☐	5870735	February 1999	Agrawal et al.	707/3
☐	5890147	March 1999	Peltonen et al.	707/1
☐	5895470	April 1999	Pirolli et al.	707/102
☐	5924090	July 1999	Krellenstein	707/5
☐	5948058	September 1999	Kudoh et al.	709/206
☐	5953718	September 1999	Wical	707/5
☐	5963940	October 1999	Liddy et al.	707/5
☐	5999925	December 1999	Evens	707/5
☐	6006221	December 1999	Liddy et al.	707/5

OTHER PUBLICATIONS

Liddy, E. D. and Myaeng S. H. "DR-LINK System: Phase I Summary" Proceedings of the TIPSTER Phase I Final Report, Sep. 1993 (published 1994), pp. 93-110.

ART-UNIT: 277

PRIMARY-EXAMINER: Choules; Jack

ATTY-AGENT-FIRM: Thompson Hine & Flory LLP

ABSTRACT:

An information management, retrieval and display system searches through an informational resource, such as a document (e.g., a treaty), a number of individual documents (e.g., Web pages resident on the Internet), or a stream of information (e.g., DNA code, source code, satellite data transmissions, etc.) and displays the results of the search in an collapsible/expandable format based upon a user-selected display criteria or hierarchy. Such a display hierarchy will allow the end-user to effectively and quickly obtain items of interest from the search results. Generally, the system performs a method for retrieving information from an informational resource that includes the steps of: (a) dividing the informational resource into a plurality of finite elements; (b) assigning a categorical tag to each of the plurality of finite elements, where the categorical tag includes data pertaining to a content of the finite element; (c) generating a searchable database record for each of the plurality of finite elements, where each searchable database record includes at least one string contained within the finite element, where the string can be a word, a phrase, a symbol, a group of symbols, a data segment or the like; (d) supplying a search string; (e) searching the searchable database for searchable database records containing the search string; (f) arranging the results of the searching step in a hierarchal structure according, at least in part, to the data in the categorical tags assigned to the finite elements found in the searching step; and (g) displaying the results of the searching step in the hierarchal structure.

h e b b g e e e f c e ef g e ge

32 Claims, 7 Drawing figures

Previous Doc Next Doc Go to Doc#

h e b b g e e e f c e ef g e ge

L1: Entry 2 of 3 File: USPT Sep 26, 2000

US-PAT-NO: 6125352
DOCUMENT-IDENTIFIER: US 6125352 A
**** See image for Certificate of Correction ****

TITLE: System and method for conducting commerce over a distributed network

DATE-ISSUED: September 26, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Franklin; D. Chase	Seattle	WA		
Remington; Darren B.	Issaquah	WA		
Saliba; Bassam	Kirkland	WA		
Speelpenning; Bert	Kirkland	WA		
Cockrill; Michael	Issaquah	WA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Microsoft Corporation	Redmond	WA			02

APPL-NO: 08/ 748688 [PALM]
DATE FILED: November 13, 1996

PARENT-CASE:
PRIORITY This application claims priority from the provisional patent application No. 60/020,891 mailed Jun. 28, 1996, titled, "SYSTEM AND METHOD FOR CONDUCTING COMMERCE OVER A DISTRIBUTED NETWORK."

INT-CL: [07] G06 F 17/60

US-CL-ISSUED: 705/26; 705/27, 709/217, 709/218, 709/219
US-CL-CURRENT: 705/26; 705/27, 709/217, 709/218, 709/219

FIELD-OF-SEARCH: 705/26, 705/27, 705/16, 705/17, 705/18, 705/1, 380/24, 380/25, 235/383, 340/825.35, 395/200.47, 395/200.48, 395/200.49, 709/218, 709/219, 709/217

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4799156	January 1989	Shavit et al.	364/401

h e b b g e e e f c e ef g e ge

☐	4992940	February 1991	Dworkin	364/401
☐	5469206	November 1995	Strubbe et al.	348/7
☐	5590197	December 1996	Chen et al.	380/24
☐	5640193	June 1997	Wellner	348/7
☐	5664110	September 1997	Green et al.	705/26
☐	5664115	September 1997	Fraser	705/37
☐	5671279	September 1997	Elgamal	380/23
☐	5677955	October 1997	Doggett et al.	380/24
☐	5710887	January 1998	Chelliah et al.	705/26
☐	5721832	February 1998	Westrope et al.	705/27
☐	5744787	April 1998	Teicher	235/380
☐	5757917	May 1998	Rose et al.	380/25
☐	5850446	December 1998	Berger et al.	380/24
☐	5918213	June 1999	Bernard et al.	705/26
☐	5956483	September 1999	Grate et al.	709/203

OTHER PUBLICATIONS

General overview and description of eShop Technology, Internet address: http://www.eshop.com/corp/technology.html. This reference was copied from the Internet and printed around May, 1996, although the pages are dated Jan. 1, 1996. A compilation of press releases of various dates describing features of eShop Technology, Internet address: http://www.eshop.com/corp/press.html. This reference was copied from Internet and printed around May, 1996, although the pages are dated Jan. 1, 1996. Also note dates listed for press release of Nov. 7, 1995, Dec. 7, 1995, and Jan. 23, 1996.

ART-UNIT: 271

PRIMARY-EXAMINER: Cosimano; Edward R.

ATTY-AGENT-FIRM: Lee & Hayes, PLLC

ABSTRACT:

A system and method for conducting commerce over a distributed network manage merchant and product information in an electronic shopping basket, payment source information in an electronic wallet, and shipping address information in an electronic address book, all of such information being stored on a consumer computer. A commerce client running on the consumer computer is configured as a MIME handler and extends the functionality of a standard Web browser to support computer-based shopping. A merchant site Web server provides HTML-coded Web documents which describe merchant products and which host computer-based shopping options. The HTML-coded Web documents contain function-calling information by which consumer-selected options invoke shopping-related functions on either the merchant (server) computer or the consumer (client) computer. A consumer selects the options from within the Web browser to initiate shopping-related operations such as: retrieve product information from merchants on the World Wide Web, selectively store product information locally on the consumer computer, locally compare product information from different merchants, locally store payment source and shipping

address information and selectively forward such information to merchant sites, order products from Web-based merchants, track the status of purchase orders, and receive instructional information on application usage.

25 Claims, 12 Drawing figures

Previous Doc Next Doc Go to Doc#

h e b b g e e e f c e ef g e ge

First Hit Fwd Refs Previous Doc Next Doc Go to Doc#
End of Result Set

Generate Collection Print

L1: Entry 3 of 3 File: USPT Feb 22, 2000

US-PAT-NO: 6029165
DOCUMENT-IDENTIFIER: US 6029165 A
** See image for Certificate of Correction **

TITLE: Search and retrieval information system and method

DATE-ISSUED: February 22, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Gable; Gary A.	Port Charlotte	FL		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Arthur Andersen LLP	Chicago	IL			02

APPL-NO: 08/ 967775 [PALM]
DATE FILED: November 12, 1997

INT-CL: [07] G06 F 17/30

US-CL-ISSUED: 707/3; 707/1
US-CL-CURRENT: 707/3; 707/1

FIELD-OF-SEARCH: 707/3, 707/1, 707/4, 707/5, 707/104

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	5659742	August 1997	Beattie et al.	707/104
☐	5675788	October 1997	Husick et al.	707/104
☐	5717914	February 1998	Husick et al.	707/5
☐	5737734	April 1998	Schultz	707/5
☐	5742816	April 1998	Barr et al.	707/104
☐	5873076	December 1998	Barr et al.	707/3
☐	5920858	July 1999	Kitabayashi et al.	707/4

h e b b g e e e f c e ef g e ge

OTHER PUBLICATIONS

Dunlap, C., "Informix releases new version of Web development tool", Information Week, Jun. 17, 1996.
Higgins, K., "Your Agent is Calling--They may still look like Web browsers, but agents are gaining sophistication and coming your way", Communications Week, Aug. 5, 1996.
Jacobs, I., "A World of Choices--Oracle, Informix and Sybase offer ways to Web-enable databases", Information Week, Jun. 15, 1996.
Marshall, M., "Oracle Goes Modular--`Cartridges` ease development of custom applications", Information Week, Sep. 30, 1996.
Rodriguez, K., "Excalibur Advances retrieval", Communications Week, Nov. 6, 1995.
Rodriguez, K., "Searching the Internet--Demand for Internet file retrieval fuels growth of search engine market", Interactive Age, Jul. 31, 1995.
Smith, T., "`MERGER`: Informix, Illustra team up--Universal Database On Its Way", Information Week, Feb. 19, 1996.

ART-UNIT: 277

PRIMARY-EXAMINER: Lintz; Paul R.

ASSISTANT-EXAMINER: Shah; Sanjiv

ATTY-AGENT-FIRM: Merchant & Gould P.C.

ABSTRACT:

A system and method for search and retrieval of electronic objects, the objects including electronically encoded information. The system and method use an electronic lexicon which is configured to provide predefined search elements that are designed to identify objects relevant to a specific community. Format filter modules identify a format of an electronic object to be searched and enable the search using the search elements within the lexicon.

30 Claims, 12 Drawing figures

Previous Doc Next Doc Go to Doc#

Using Micro Information Units for Internet Search

Xiaoli Li
School of Computing
National University of Singapore
3 Science Drive 2
Singapore 117543
lixl@comp.nus.edu.sg

Bing Liu
Department of Computer Science
University of Illinois at Chicago
851 S. Morgan Street
Chicago, IL 60607-7053
liub@cs.uic.edu

Tong-Heng Phang, Minqing Hu
School of Computing
National University of Singapore
3 Science Drive 2
Singapore 117543
phangth, humq@comp.nus.edu.sg

ABSTRACT

Internet search is one of the most important applications of the Web. A search engine takes the user's keywords to retrieve and to rank those pages that contain the keywords. One shortcoming of existing search techniques is that they do not give due consideration to the micro-structures of a Web page. A Web page is often populated with a number of small information units, which we call *micro information units* (MIU). Each unit focuses on a specific topic and occupies a specific area of the page. During the search, if all the keywords in the user query occur in a single MIU of a page, the top ranking results returned by a search engine are generally relevant and useful. However, if the query words scatter at different MIUs in a page, the pages returned can be quite irrelevant (which causes low precision). The reason for this is that although a page has information on individual MIUs, it may not have information on their intersections. In this paper, we propose a technique to solve this problem. At the off-line pre-processing stage, we segment each page to identify the MIUs in the page, and index the keywords of the page according to the MIUs in which they occur. In searching, our retrieval and ranking algorithm utilizes this additional information to return those most relevant pages. Experimental results show that this method is able to significantly improve the search precision.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval – *Retrieval models, Search process.*

General Terms

Algorithms, Experimentation.

Keywords

Web Search, Micro Information Units, Web page segmentation.

1. INTRODUCTION

One of the most important applications of the Web is the search using search engines, e.g., AltaVista, Google, Yahoo, etc. These search systems allow the user to specify some keywords to retrieve those Web pages that contain the keywords. A major shortcoming of the current techniques is that they do not consider different topic areas of a page. Typically, the contents of a Web page encompass a number of related or even unrelated topics. Each topic usually occupies a separate area in the page. We call each topic area a *micro information unit* (or MIU in short). For example, a bookstore Web page selling books may include other diverse information like stock market quotations and weather forecasting. A personal homepage may contain information on different interests of its owner.

A *micro information unit* (MIU) is a coherent topic area according to its content, and it is usually also a visual block from the display point of view. If the user's query terms (or keywords) occur in a single MIU of a Web page, the pages returned by a search engine are generally relevant and useful. However, if the keywords scatter at different MIUs, it can cause low precision of the returned search results. Although many search engines are able to consider relative distances of keywords [4] (among others, e.g., word frequency, authority and hub scores, etc) in a Web page in their ranking processes, they do not consider whether these words occur in different MIUs or a single MIU.

Let us use an example to illustrate the problem. For instance, we wish to find some free downloadable videos. We issue the search query "free download video" to the search engine Google. Google returns a large number of Web pages. However, most top ranking pages do not offer any free downloadable videos. For example, the first page returned by Google does contain the three keywords "free", "download" and "video" (Figure 1). It is a site that sells software for playing audio and video. It does not have any free video for downloading. From Figure 1, we observe that "free", "download" and "video" (circled in the figure) appear in different MIUs or topic areas.

In this paper, we propose a technique to deal with the problem. The key idea is to segment each Web page to identify different micro information units or topic areas according to its HTML tags and contents. In searching, if the keywords of a query occur in the same MIU, the Web page will be given a higher ranking score. Otherwise, it will be given a lower ranking score. In the proposed technique, page segmentation and indexing according to MIUs in a Web page is done in off-line pre-processing. We show that the additional information on MIUs can be naturally integrated with inverted lists indexing commonly used by Web search engines. In on-line search, our retrieval and ranking algorithm makes use of this MIU information to sort the relevant pages. Due to seamless integration of MIUs with inverted lists, additional computation required during searching is minimum.

CIKM'02, November 4-9, 2002, McLean, Virginia, USA.

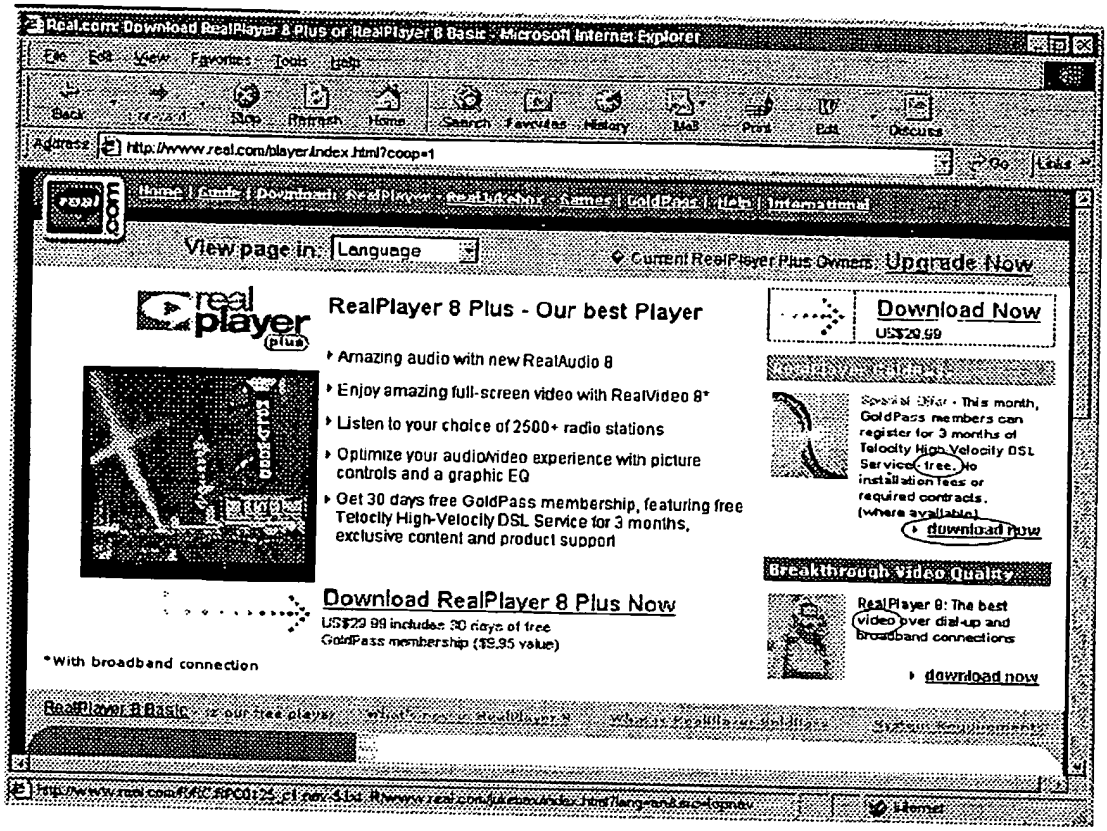

Figure 1: The first page from Google for the query, "free download video"

The proposed technique is intended to be used as an advanced search option for a search engine (which we also call the *base search engine*). That is, when the precision of the results returned by the base search engine is low, we can employ the proposed technique to re-rank the results.

To evaluate the proposed technique, we used Google as the base search engine. When the precision of the returned results by Google is low, we re-rank its top 200 pages. Experimental results (including comparison with Google) show that our method is able to improve the search precision dramatically, i.e., after re-ranking the number of relevant pages at the top of the list increases significantly.

2. RELATED WORK

The key issue in Web search is how to efficiently retrieve relevant Web pages with high precision for its top ranking results. The main technique used in current search engines is keyword matching. In recent years, a number of works were reported to improve search by using additional information from Web pages. [4] presents the Google search engine, which employs link structures and anchor text in addition to the traditional factors such as word occurrence and frequency to make relevance judgments. [18] presents a similar link-based search method. [16] examines the actual pages suggested by multiple search engines and then displays the results according to the user's query. Clever search engine [6] incorporates several algorithms (i.e. HIT algorithm) that make use of hyperlink structure for discovering high-quality information on the Web. [9] describes a Web searching method where the input is the URL of a page. It only uses the connectivity information to identify related pages. These works are all different from ours, as they do not segment each Web page to identify different MIUs and then use these MIUs to aid the search.

[26] presents an algorithm to efficiently retrieve information units, which are logical Web document consisting of multiple physical pages. Our micro information unit is a topic area within a physical Web page.

[5] uses the Document Object Model (or HTML tag tree) and hyperlinks for topic distillation. It segments the HTML tag tree for the purpose of computing authority and hub scores of the intermediate subtrees in relation to other pages and links. This is different from our work as we aim to find coherent topic areas of the current page using both text contents and display properties.

Segmentation of text documents has been studied extensively in information retrieval. Existing techniques roughly fall into two categories: lexical cohesion methods and multi-source methods [7]. The former identifies coherent blocks of text with similar vocabulary [2, 10, 15, 23]. The latter combines lexical cohesion with other indicators of topic shift, such as relative performance of two statistical language models and cue words [1, 3]. In [11], Hearst discussed the merits of imposing structure on full-length

text documents and reported good results of using local structures for information retrieval.

There are also several works on passage retrieval method although mostly applied in the area of text retrieval [14, 21]. [24] proposes sub-document access: it allows the user to zoom in on parts of the full text that are meaningful to his task. [8] utilizes the reciprocal of the length of each passage as an estimate of its relevance. [20] discusses that the query occurs in unrelated context results in non-relevant document return.

Our Web search based on MIUs is different from the research above. The nature of Web pages differs from a static text document. *Web contents* can switch from one topic to a completely different topic abruptly without requiring additional textual cues to "bridge the topic shift". Gradual topic shifts in text documents are often indicated by certain textual cues (e.g., "next, consider...", "firstly... secondly..."). Such cues, employed by text segmentation, are not applicable to Web pages. In our case, we considered the changes in visual cues (e.g., bold emphasis, sudden increase in font-size, change in font-color) of Web pages. Visual cues offer indications that a topic may have shifted within a Web page. In our Web page segmentation, we make use of both contents and presentation styles or visual features of the Web page to segment the page.

3. PRE-PROCESSING WEB PAGES

We now present the proposed technique. This section focuses on pre-processing of each Web page, i.e., building a HTML tag tree and segmenting the Web page into MIUs using the tag tree. The next section describes our ranking algorithm. All the procedures discussed in this section are done off-line. Since the proposed technique is used as an advanced search method or option for a base search engine, all the required information is assumed to be stored at the base search engine site.

3.1 Building HTML Tag Trees

Web pages are hypertext documents written in HTML that consists of plain text, tags and links to image, audio, and video files, etc. Like most search engines, our technique only uses plain text and tags in search. Plain text are strings of characters not embedded within any tags. It can have different appearances in terms of color, font, size and style as specified by tags. Tags (enclosed by a pair of angular brackets) define the display properties and characteristics of a Web page. In general, most Web documents are constituted of *opening* and *closing* pairs of HTML tags (indicated by < > and </> respectively). Within each corresponding tag-pair, it can contain other pairs of tags, resulting in nested blocks of HTML codes.

Based on the nature of nested structure of HTML codes, a *Tag Tree* can be built in a fairly straightforward manner for each Web page using its HTML source. A *node* in the tag tree contains a tag name, content text and its display attributes (color, font, size, etc).

3.2 Segmenting the Tag Tree into MIUs

We now segment the Web page into various MIUs using the tag tree. Although the tag tree already gives us an initial segmentation of the page, it is often too refined and is solely based on presentation features of the page. We need to merge some nodes in the tree to form coherent topic or information units. Our segmentation technique is based on both content and display similarities.

Merging of nodes is done in two steps: (1) merging each heading and its immediate content paragraph (note that a content paragraph may not have the and tags); (2) merging two adjacent text paragraphs. Below, we discuss these steps in turn.

Step 1 - ***Merging each heading and its immediate content paragraph***: In this step, we scan all the sibling nodes of a sub-tree from left to right to find all heading and paragraph pairs. This is performed in 2 sub-steps:

(i) *Identifying all potential heading and content paragraph pairs*: Let A and B be any two different leaf nodes of a sub-tree. We use *Len* to denote the length (number of words) of the text string stored in A or B. We use *tagRank* to denote the *font emphasis* given to the text strings stored in A or B. The value of *tagRank* is based on the *priority*. The highest value of *tagRank* is assigned to header tags (e.g..: <h1>, <h2> and so on), followed by formatting tags (e.g. <b>, , <blink>) and enlarged font sizes (<big>, <size...>). All the other tags are assigned the same rank value that is lower than the three types above. In general, a paragraph heading tends to be more prominent and distinct in terms of font size or appearance as compared to its content paragraph.

We use $Neig(A, B)$ to denote the neighboring-relation of A and B, and node A is the left neighbor of B. The following condition is used to determine whether A is a potential heading for B (or B is A's immediate content paragraph). $((A \cap B) \neq \emptyset)$ means at least one word (term) in A (node A) also occur in its immediate text paragraph B).

$$(tagRank(A) >= tagRank(B)) \wedge (Len(A) < Len(B)) \wedge Neig(A, B) \wedge (A \cap B) \neq \emptyset) \quad (1)$$

Here, we use the length of A and B, font size attributes of A and B, and their neighborhood relation to check whether A is potentially a heading for B. Note that this is computed after *stop-words elimination* and *word stemming* have been performed. We use the Porter's algorithm given in [22] for the purposes.

(ii) *Further evaluation*: After (i), we have identified all the potential pairs. This sub-step further evaluates them using their display properties. For each (A, B) pair, we try to find the next pair (C, D) which also has a possible heading and content paragraph relationship as computed in sub-step (i). We then evaluate A, B, C and D using the display similarity, *DisplaySim*. *DisplaySim* counts the number of identical features (display properties) of any two nodes. We use the following condition:

$$\exists(C, D), Neig(C, D) \wedge (DisplaySim(A, C) >= \delta) \wedge (DisplaySim(B, D) >= \delta) \quad (2)$$

where $DisplaySim(X, Y) = |X.features \cap Y.features|$ (which is the size of the intersection). The set of features includes font, size, color, tag name, and default. We set $\delta = 3$ (determined from experimental observations), which means if $(DisplaySim(A, C) >= 3)$ we consider they have high display similarity (this also applied to B and D).

This condition basically tries to see whether A and B have a parallel pair (C, D). If so, we confirm the heading and content paragraph relationship of A and B, and that of C and D. We

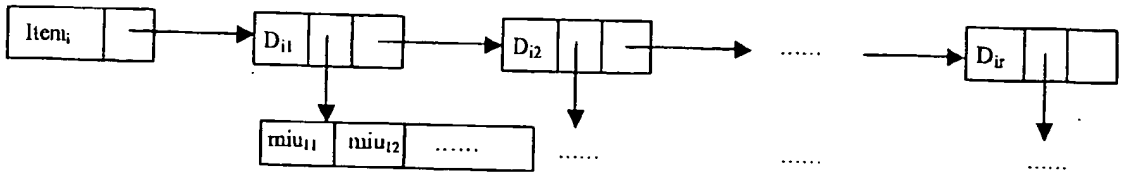

Figure 3: An inverted list with data structures for MIUs

believe that the display property comparison is more meaningful here since people often are able to segment a Web page correctly even they do not know the content of the page.

If conditions (i) and (ii) are both satisfied, we merge nodes A and B, and at the same time C and D, i.e., to put the attributes of B into A, and the attributes of D into C. Nodes B and D are deleted.

Step 2 - *Merging two adjacent text paragraphs*: Here, we wish to join similar text paragraphs (some paragraphs may contain their headings after step 1). Let X and Y be two text paragraph nodes within the same sub-tree. We now compute their degree of content similarity, *ContentSim*(X, Y). The *inner product* [17] is employed for the purpose (m is the total number of terms or keywords in $X \cup Y$). If term i exists in X, then $x_i = 1$, otherwise $x_i = 0$. If i exists in Y, $x_i' = 1$, otherwise $x_i' = 0$.

$$\text{ContentSim}\ (X, Y) = \sum_{i=1}^{m} x_i x_i' \qquad (3)$$

If $ContentSim(X,Y) \geq \varpi$, we say that nodes X and Y have a high similarity. We can combine their contents, i.e., placing the content of Y into X. We set $\varpi = 2$, determined by experiments. It reflects the acceptable level of similarity among various nodes well.

The overall algorithm is given in Figure 2. *maxDepth* is the maximum depth of the original tag tree. *treeDepth* is the depth of the tree that is being worked on. *Stree* is the set of all sub-trees at depth *treeDepth*. Each *subtree*$_i$ only contains leaf nodes, and no sub-trees below.

```
1  for (treeDepth = maxDepth −1; treeDepth < 0; treeDepth--) do
2    Stree = {subtree_i | subtree_i is a sub-tree at level treeDepth};
3    while |Stree| > 0 do /* |Stree| is the size of the set Stree */
4      for each subtree_i ∈ Stree do
5        for each Neig(A, B) do
6          if conditions (1) are satisfied then
7            if ∃pair(C, D) & conditions (2) is satisfied then
8              Merge node A and B;
9        endfor
10     endfor
11     for each subtree_i ∈ Stree do
12       Scan all the nodes and their sibling nodes;
13       if ContentSim(X, Y) ≥ ϖ ,then
14         Merge node X and Y;
15     endfor
16     for each subtree_i ∈ Stree do
17       If A has no sibling then
18         move its content into its parent and delete it
19     endfor
20   endwhile
21 endfor
```

Figure 2: Merging nodes of a tag tree (segmenting a page)

4. THE RANKING ALGORITHM

After obtaining the MIUs from each page through segmentation, we index the Web pages in such a way that they can be retrieved and ranked quickly. As in normal search, we also use inverted lists to store the information of the Web pages. Thus, the search technique we adopted is similar to those in a normal search engine [4]. The main difference is that in our technique we need to index and retrieve MIUs of each page. We simply add an extra data structure to each inverted list node to indicate in which MIUs each word appears. Figure 3 illustrates the inverted lists indexing with the data structures for MIUs:

Here *Item*$_i$ (i =1, 2, ...) is a word, D_{ij} is the j-th document that *Item*$_i$ occurs in and MIU$_{xy}$ is the y-th MIU in the x-th subtree of the page D_{ij}. Each node includes three fields: *ID* (document *ID*), *seg* (a pointer to all the MIUs of the page containing *Item*$_i$), *next* (a pointer to next page). Note D_{ij} in an inverted list is stored in the increasing order. For any user query, $Q = \{k_1, k_2 \ldots\ldots, k_n\}$, we will consider if the words occur in the same or neighboring MIUs in the same subtree of the same page.

A search engine typically considers many factors in its ranking algorithm, e.g., *hyperlink information* (such as *authority score* and *hub score*), *word count-weight*, *type-weight* (title, anchor, URL, font size, etc), and *type-prox-weight* (how close multi-words occur in every type) [4]. In our ranking algorithm, we only focus on whether the query terms occur in a single MIU (or 2 neighboring MIUs within the same sub-tree) of a page. Since the proposed technique is intended to be used as an *advanced search* method for a *base search engine*, we utilize our MIU-based information and also the ranking information from the base search engine in our ranking process. The reason that we need ranking information from the base search engine is because we do not need to consider other factors except our MIU-based factor in our ranking algorithm. However, since we do not have access to any existing search engine program, only the ordering information of the pages returned by the base search engine is employed in our current ranking algorithm. If a search engine system is available, all factors should be integrated in a more sophisticated manner. Section 5 shows that even this simple approach is already able to produce remarkably good results.

The proposed ranking method aims to re-rank the results returned by the base search engine when the precision of its results is poor. The number of pages to be re-ranked is specified by the user. In our experiments, we re-rank the first 200 pages from Google. Re-ranking is done on-line at query time. Pre-processing as discussed in Section 3 is done off-line for all the pages at the search engine site, as it is not possible to know what queries will be issued by users, and it is too slow to do pre-processing of the top ranking pages from the base search engine at query time.

Our ranking algorithm basically computes two scores for each page, a *primary score* and a *secondary score*. The primary score is the maximum number of query terms that occur in a MIU of the page p. Let seg_i be the terms contained in i-th MIU of p, and *queryTerms* be the set of terms in the user query. The primary score of page p (denoted by $prScore(p)$) is computed as follows:

$$prScore(p) = \arg\max_i(|\, seg_i \cap queryTerms\,|) \quad (4)$$

If the primary score of page p is less than the number of query terms (i.e., not all query terms are covered), we compute the secondary score, which takes into account of the neighboring MIU on the right of each MIU in the same sub-tree. Let seg_{ji} be the set of terms in the ith MIU of the sub-tree j. The secondary score of p (denoted by $seScore(p)$) is computed with:

$$seScore(p) = \arg\max_j(\arg\max_i(|\,(seg_{ji} \cup seg_{j,(i+1)}) \cap queryTerms\,|)) \quad (5)$$

The overall ranking algorithm is given in Figure 4.

1 Create a set of variables $pageSet_i$, $i = 1, \ldots, n$;
2 $pageSet_i = \emptyset$;
3 **for** all $p \in AllPages$ **do** $prScore[p] = 0$, $seScore[p] = 0$;
4 Retrieve the inverted lists of the query words $k_1, k_2, \ldots\ldots, k_n$;
5 Initialize pointer set: $L = \{p_1, p_2, \ldots, p_n\}$, here each p_i point to the first node in the corresponding link list;
6 **while** $\exists p_i \neq$ Nil, $p_i \in L$ **do**
7 $md = \min(p_i.id)$, $p_i \in L$;
8 Construct a pointer set LS from L: $\{p_j | p_j.id = md\}$;
9 **if** $|LS| = 1$ **then**
10 $prScore[p_j.id] = 1$, $seScore[p_j.id] = 1$;
11 else scan all the MIUs to compute *prScore* and *seScore* of the page by checking if the query words occur in the same or neighboring MIUs.
12 **for** all $p_j \in LS$ **do** $p_j = p_j.next$;
13 **endwhile**;
14 **for** each $p \in AllPages$ **do**
15 **if** $prScore(p) = n$ **then** $pageSet_n = pageSet_n \cup \{p\}$;
16 **else if** $seScore(p) = i$ **then** $pageSet_i = pageSet_i \cup \{p\}$;
17 **endfor**
18 Rank pages in the order of $pageset_n$, $pageset_{n-1}$ and so on. For the pages in each $pageset_i$, we follow their relative ranking in the results produced by the base search engine;

Figure 4: The ranking algorithm

In Figure 4, n is the number of query terms. *AllPages* is the set of top ranking pages (to be re-ranked) from the base search engine. Lines 1 and 2 create and initialize a set of set-variables to store the resulting pages as the first level ranking (which will become clear below). Line 3 initializes two arrays used to store the final Web page scores. Given user's query, lines 4 and line 5 retrieve the inverted lists of the query words and then create a pointer set. Each pointer in the pointer set points to the first Web page of an inverted list. From line 6 to line 13, we compute *prScore* and *seScore* for all the Web pages. The loop ends when all the pointers reach the end of the inverted lists, which means we have already finished processing all the Web pages in the inverted lists. In each loop, for all retrieved inverted lists, we first find the page with smallest document *ID* (*md*). After we process it (give this page the *prScore* and *seScore* scores), we move the pointers that point to the smallest document *ID*s to the next node to begin the next loop. In a loop, if the page contains only one query word, both *prScore* and *seScore* are given the score of 1. Otherwise, the page contains at least two words in the user's query. Then, we need to check if they occur in the same or neighboring MIUs. In this process, we update the maximal number of query words contained in a single MIU and two neighboring MIUs. In line 15, if $prScore(p) = n$, p should be one of the top ranking pages, stored in $pageSet_n$ (since we believe that a MIU in a Web page that contains all the query terms is very likely to be relevant to the user). If $prScore(p)$ is less than n, we store p into $pageset_{n-i}$ according to its *seScore*, where n-i indicates how many keywords are found in two neighboring MIUs (line 16). Finally, we have a two-level ranking (line 18). The first level ranks the sets of pages in the order of $pageset_n$, $pageset_{n-1}$ and so on. The second level ranks all the pages in each $pageset_i$ according to their relative ranking in the results produced by the base search engine.

The complexity of our algorithm is similar to the complexity of a normal search engine. In a normal search engine, given a query, its main task is to check if the query terms occur in the same page, so the complexity is $|query|*v$ on average (we ignore the other cost in computing *authority score, hub score, word count* etc). Here $|query|$ is the number of query words, and v is the average length of all inverted lists. In our algorithm, we also need to check in each page whether the query words occur in the same or neighboring MIUs. Thus, it needs to traverse the MIU list of each page. Then complexity of our algorithm is $|query|*v*q$. Here q is the average number of MIUs in all the pages. Since q is normally very small, thus little extra time is needed by our new search technique.

5. EXPERIMENTAL RESULTS

This section evaluates the proposed technique. We first compare the precision results of our method with those from Google, and then discuss its running efficiency.

Evaluation of the ranking effectiveness is difficult in the context of web search because of the difficult tasks in (i) *choosing queries* and (ii) *evaluating the relevance* of search results. Our criteria for *choosing queries* are: they should be from diverse areas and unambiguous. By unambiguous, we mean that the intent of each query is agreed upon by a panel of 3 judges. We used queries from two independent sources, the entire collection of queries (351-400) from TREC-7 [25], and 30 queries from Metaspy of MetaCrawler [19] (which allows users to view others' queries being submitted to the system). For queries from Metaspy, we first collected a list of continuous queries and then removed those queries that are ambiguous, i.e., our panel of judges could not decide the intension of the user.

As for *evaluating the relevance* or correctness of the search results, the web pages produced should satisfy the conditions pre-defined by our judges or correspond to the standard narratives provided by TREC [25]. For example, TREC Query 354: Journalist Risks, the narratives stated are "*any document identifying an instance where a journalist has been killed, arrested or taken hostage in the performance of his work is relevant.*" Our judges evaluate the relevance of the search results with such narratives to obtain a consensus on the search precision.

The choice of using Google as a basis for re-ranking (*base search*

engine) is because of its state-of-the-art search mechanism. In general, Google performs very well as a general-purpose search engine. However, there exist query phrases that it fails to perform satisfactorily. Our purpose is to provide advanced re-rankings for queries whose Google's precisions are low. For each query, we re-rank the first 200 search results from Google, after crawling and pre-processing the pages.

In general, information retrieval systems are evaluated using both precision and recall measures. However, in the context of Web search, the precision of the top-ranking results returned by a search engine is more important since most people only see the top 20-30 results [12, 13]. That is, even if a search engine has high recall, but if most of the relevant results are located below 20-30 top ranking results, there is little chance that the user will see them. Thus, many researchers believe that high precision is important even at the expense of recall [4]. In our experiments, we are only using precision of top 20 ranking results to evaluate the performance of our system.

The precisions of the top 20 ranking results from Google and our method (MIU) are compared in Table 1. The first column states the source of queries. The second and third columns list the average precisions of the top 20 results from Google and our method respectively. The fourth column provides the improvement percentage of MIU over Google on the average precision. Tables 3 and 4 in the Appendix list all the search queries and the corresponding precision for both data collections.

Table 1. Average precision comparison for TREC-7 and MetaSpy

Data Collections	Average Precision		Improvement
	Google	MIU	
TREC-7 (351-400)	0.50	0.59	18.00 %
MetaSpy	0.54	0.63	16.67 %

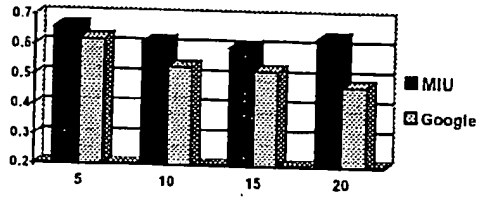

Figure 5: Average precision comparison per 5 returned pages of MIU and Google for TREC7 queries

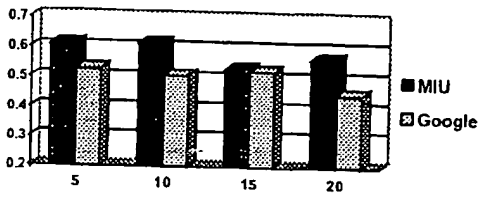

Figure 6: Average precision comparison per 5 returned pages of MIU and Google for MetaSpy queries

From Table 1, we observe that the average precision after our re-ranking is substantially higher. The improvement in precision by our system over that of Google is 18% for TREC-7 queries and 16.67% for Metaspy queries. Figures 5 and 6 give the graphical comparison of the average precision of every 5 returned pages for both data collections. We observe that in general MIU is superior to Google for any number of top-ranked pages (used in computing precision).

Table 2 presents the won-lost-tied record of MIU against Google. For the Trec-7 data collection, 58% of the total number of queries increased in precision; 20% of queries remained unchanged and 22% of queries decreased in precision after applying our MIU method as compared to Google's ranking results. For the MetaSpy query collection, the precisions of 67% of the queries increased; the precisions of 3% of the queries remain unchanged and 30% of queries decreased. We observe that most instances of MIU performing worse than Google occur when the precisions of Google's results tend to be rather high. For example, for those queries that Google has better results, its average precision is 0.73 for the MetaSpy data collection. This precision value of Google should be highly satisfactory for most users and does not require additional MIU processing. That is why we say that our MIU method can be seen as an advanced search option. It should be used when Google's results are not satisfactory.

Table 2: Won-lost-tied record for TREC-7 and MetaSpy queries

Data Collections	Increase	Draw	Decrease
TREC-7 (351-400)	58 %	20 %	22 %
MetaSpy	67 %	3 %	30 %

We now briefly discuss the running efficiency of our system. We use a single machine (Sun E450 250MH with 500MB memory and a single processor) for all our experiments. In pre-processing, the major operations involved are crawling and indexing. It is difficult to measure how long crawling took overall because of complications like bandwidth limitations, crashed name servers, congested network and others. For indexing, the indexer runs at roughly 4 pages per second. Our indexer is not running in parallel, which affects the speed. All these pre-processing are mostly duplicated works of Google. They can be easily incorporated into Google, which will improve the performance significantly. For ranking, our ranking procedure handles roughly 50 pages per second, or 2 to 10 seconds for each query (which is mostly dominated by disk IO). Improving the efficiency of crawling, indexing and searching was not the main focus of this research. With further optimization and more powerful machines, the running speed can be improved significantly.

6. CONCLUSION

In this paper, we presented a technique to improve the precision of Web search. It is based on the idea of segmenting each web page into different MIUs (topic areas) according to its contents and HTML tags. In searching, only the terms in a single unit or at most two neighboring units of a page are used to match the user's query terms. This is different from existing techniques used by current search engines, which typically employ all the terms in the whole page to match the query terms. From the experiment results shown in Section 5, we observe that the precision of the ranking produced by our method is substantially higher.

7. REFERENCES

[1]. J. Allan, J. Carbonell, G. Doddington, J. Yamron, and Y. Yang. "Topic detection and tracking pilot study final report." *DARPA Broadcast News Transcription and Understanding Workshop*, 1998.

[2]. D. Beeferman, A.Berger, and J. Lafferty, "A model of lexical attraction and repulsion." *ACL-97*, 1997.

[3]. D. Beeferman, A. Berger & J. Lafferty. "Statistical models for text segmentation." *Machine learning*, 34(1-3), 1999.

[4]. S. Brin, L. Page, "The anatomy of a large-scale hypertexture Web search engine." *Computer Networks* 30 (1-7), 1998.

[5]. S. Chakrabarti. "Integrating the document object model with hyperlinks for enhanced topic distillation and information extraction." *WWW10*, 2001.

[6]. The CLEVER Project, http://www.almaden.ibm.com/cs/k53/clever.html

[7]. F. Y. Choi. "Advances in domain independent linear text segmentation." *NAACL'00*, Seattle, USA, 2000.

[8]. G. V. Cormack, C.L.A. Clarke, C.R. Palmer and S.S.L. "To Passage-Based Refinement (MultiText Experiments for TREC-6)." In D. K. Harman and Ellen Voorhees, editors, *The Sixth Text REtrieval Conference (TREC-6)*, 1998.

[9]. J. Dean, M. R. Henzinger. "Finding Related Pages in the World Wide Web," *WWW8*, 1999.

[10]. D. Eichmann, M. Ruiz, and P. Srinivasan, "A Cluster-based approach to tracking, detection and segmentation of broadcast news." *DARPA Broadcast News Workshop*, 1999.

[11]. M. Hearst. "Subtopic structuring for full-length document access." *ACM SIGIR 93*, 1993.

[12]. C. Hoelshher. "How Internet experts search for information on the Web." *The World Conference of the World Wide Web, Internet and Intranet*, 1988, Orlando, FL.

[13]. B. J. Jansen, "The effect of query complexity on Web searching results," *Information Research*, 6(1), 2000.

[14]. M. Kaszkiel, J. Zobel, "Passage Retrieval Revisited." *ACM SIGIR 97*, Philadelphia, PA, USA.

[15]. S. Kaufmann. "Cohension and collocation: Using context vectors in text segmentation." ACL-99, 1999.

[16]. S. Lawrence, and L. Giles, "Context and page analysis for improved web search."*IEEE Internet Computing*,2(4) 1998.

[17]. D. D. Lewis, et al. "Training algorithms for linear text classifiers." *ACM SIGIR 96*,1996.

[18]. Y. Li. "Toward a qualitative search engine," *IEEE Internet Computing*, 1998, July.

[19]. MetaCrawler Search Engine www.metacrawler.com Metaspy www.metaspy.com

[20]. M. Mitra, A. Singhal and C. Buckley. "Improving Automatic Query Expansion." *SIGIR98*, 1998.

[21]. E. Mittendorf and P. Schuble. "Document and Passage Retrieval Based on Hidden Markov Models." *SIGIR96*.

[22]. M. Porter. "An algorithm for suffix stripping." *Program*, 14(3): 130-137, 1980.

[23]. J. C. Reynar, "Statistical models for topic segmentation." *ACL-99*, 1999.

[24]. G. Salton, J. Allan and C. Buckley, "Approaches to Passage Retrieval in Full Text Information Systems." *SIGIR-93*, 1993, pp. 49-58.

[25]. Text REtrieval Conference (TREC) Data - English Test Questions (Topics) File List http://trec.nist.gov/data/topics_eng/index.html

[26]. W. S Lee, K. S. Candan, V. Quoc and D. Agrawal. "Retrieval and organizing Web pages by Information Unit." *WWW10*, Hongkong, 2001.

APPENDIX

Table 3: Precision comparison using MetaSpy queries (the queries are ordered according to the won-tied-lost record against Google)

Search Query	Google	MIU	Search Query	Google	MIU
star wars wallpaper	0.85	1.00	free download music	0.30	0.70
Free craft projects	0.60	0.80	information history tomatoes	0.45	0.65
supermodel success stories	0.35	0.50	literary films list	0.10	0.35
laser eye surgery	0.80	0.95	Singapore programming jobs	0.35	0.65
accident death photo	0.25	0.50	red ladies t-shirt	0.40	0.55
motorcycle dealers in Texas	0.45	0.55	html tag tree	0.50	0.50
First Communion letters	0.20	0.50	crime rates and ethnicity	0.55	0.50
entertainment in San Diego	0.45	0.85	Christmas island tour	0.60	0.55
karaoke machine	0.65	0.75	Mickey mouse club	0.80	0.70
growing marijuana	0.40	0.45	internet service provider illinois	0.75	0.65
award winning web sites	0.45	0.70	studies on travel writing	0.45	0.20
gall bladder surgery causes	0.50	0.70	California legal codes	0.90	0.70
alternative music origins	0.35	0.50	plant pathology journals	0.89	0.75
decorative candlestick sale	0.40	0.80	Michael Jordan shoes	0.95	0.70
heavyweight boxing championship	0.55	0.85	organizational industrial psychology	0.70	0.40

Table 4: Precision comparison using TREC-7 (we re-order the queries in TREC-7 according to the won-tied-lost record against Google)

Search Query	Google	MIU	Search Query	Google	MIU
Falkland petroleum exploration	0.55	0.70	mercy killing	0.10	0.25
British Chunnel impact	0.25	0.50	home schooling	0.20	0.30
journalist risks	0.05	0.60	autombile recalls	0.15	0.25
postmenopausal estrogen Britain	0.10	0.11	dismantling Europe's arsenal	0.20	0.60
human smuggling	0.60	0.85	euro opposition	0.70	0.70
transportation tunnel disasters	0.30	0.40	mainstreaming	0.35	0.35
anorexia nervosa bulimia	0.70	0.80	piracy	0.15	0.15
Food/drug laws	0.70	0.75	in vitro fertilization	1.00	1.00
health insurance holistic	0.30	0.45	rabies	1.00	1.00
Native American casino	0.45	0.60	El Nino	1.00	1.00
encryption equipment export	0.70	1.00	robotics	0.45	0.45
Nobel prize winners	0.85	0.95	tourism	0.00	0.00
hydrogen energy	0.80	0.95	sick building syndrome	0.60	0.60
World Court	0.75	0.90	amazon rain forest	0.10	0.10
obesity medical treatment	0.65	0.85	ocean remote sensing	0.85	0.40
alternative medicine	0.70	1.00	territorial waters dispute	0.60	0.55
mental illness drugs	0.20	0.50	blood-alcohol fatalities	1.00	0.80
space station moon	0.15	0.55	mutual fund predictors	0.50	0.20
hybrid fuel cars	0.65	0.85	drug legalization benefits	0.90	0.79
teaching disabled children	0.45	0.50	clothing sweatshops	0.80	0.75
radioactive waste	0.45	0.55	antarctica exploration	0.70	0.53
organic soil enhancement	0.45	0.60	commercial cyanide uses	0.68	0.63
illegal technology transfer	0.30	0.45	cigar smoking	0.35	0.25
orphan drugs	0.40	0.70	hydrogen fuel automobiles	0.90	0.85
r&d drug prices	0.30	0.60	oceanographic vessels	0.20	0.15